

msdn magazine



JavaScript

Error-Handling
Techniques in WinJS.....22

Create mobile apps with HTML5, JavaScript and Visual Studio

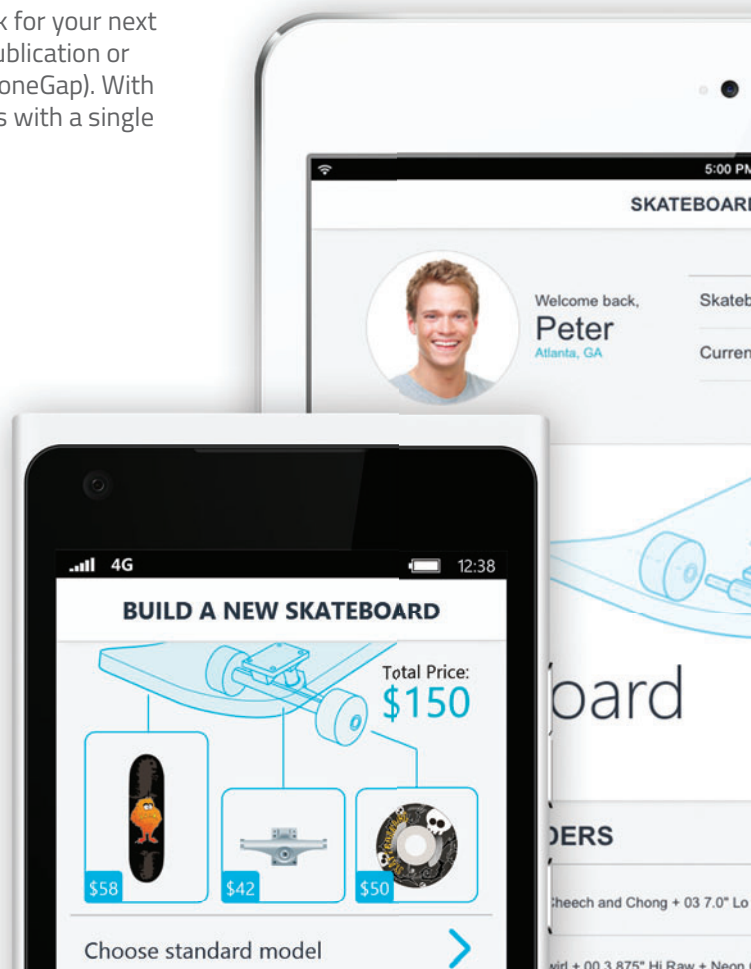
DevExtreme Mobile is a single page application (SPA) framework for your next Windows Phone, iOS and Android application, ready for online publication or packaged as a store-ready native app using Apache Cordova (PhoneGap). With DevExtreme, you can target today's most popular mobile devices with a single codebase and create interactive solutions that will amaze.

Get started today...

- Leverage your existing Visual Studio expertise.
- Build a real app, not just a web page.
- Deliver a native UI and experience on all supported devices.
- Use over 30 built-in touch optimized widgets.



Learn more and download your free trial
devexpress.com/mobile





Your next great app starts here.

From interactive Desktop applications, to immersive Web and Mobile solutions, development tools built to meet your needs today and ensure your continued success tomorrow. Download your free 30-day trial today and Experience the DevExpress Difference.



Learn more and download your free trial at devexpress.com/try

msdn

magazine



Error-Handling
Techniques in WinJS.....22

Build More Efficient Windows Store Apps Using JavaScript: Error Handling Eric Schmidt	22
Introducing Visual Studio Online Ed Blankenship	32
Build Fault-Tolerant Composite Applications Ivan Krivyakov	40
Create Modern Microfiche with the Chronicling America API Tim Kulp	50
Troubleshooting Applications with IIS Logs Eduardo Sanabria	60
Unit Testing SQL Server OLAP Cubes Using C# Mark Nadelson	64

COLUMNS

WINDOWS WITH C++

Using Regular Expressions
with Modern C++
Kenny Kerr 6

DATA POINTS

Code First Goodies
in Entity Framework 6
Julie Lerman 12

TEST RUN

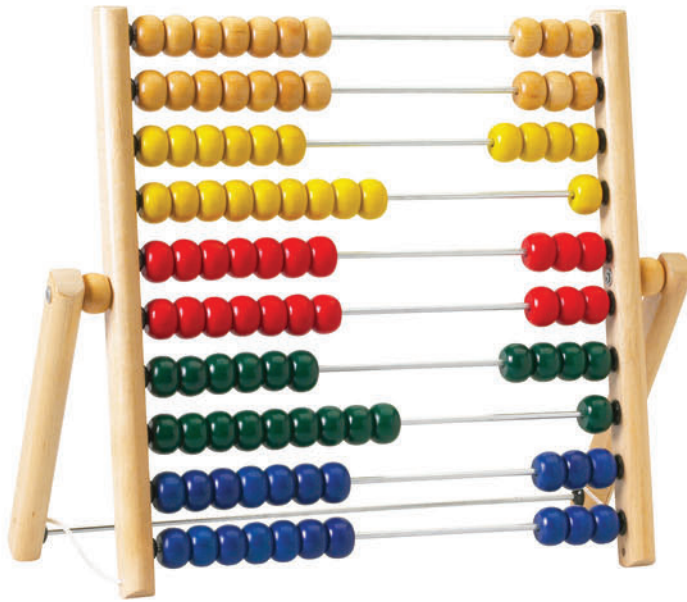
Frequent Item-Sets for
Association Rule Learning
James McCaffrey 70

THE WORKING PROGRAMMER

Getting Started with Oak:
Database Interaction
Ted Neward 76

DON'T GET ME STARTED

The Steam Drill
David Platt 80

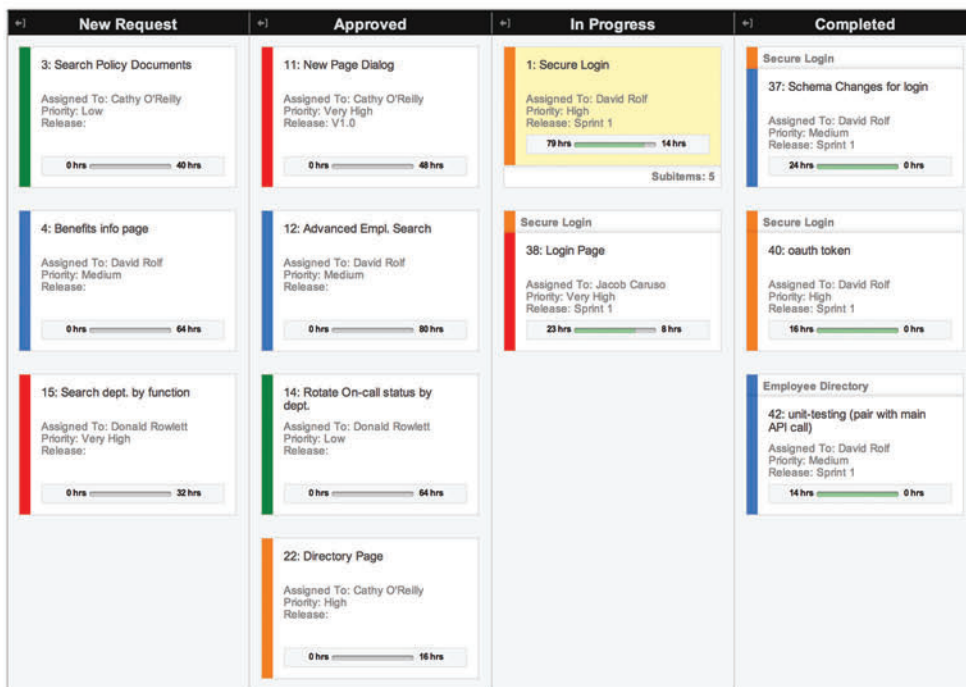


VS.





VS.



OnTime UPGRADE TO THE #1 SELLING SCRUM SOFTWARE

Technology doesn't have to make things more complicated. Step up your game and learn what real productivity feels like at OnTimeNow.com/MSDN.



dtSearch®

Instantly Search Terabytes of Text

25+ fielded and full-text search types

dtSearch's **own document filters** support "Office," PDF, HTML, XML, ZIP, emails (with nested attachments), and many other file types

Supports databases as well as static and dynamic websites

Highlights hits in all of the above

APIs for .NET, Java, C++, SQL, etc.

64-bit and 32-bit; Win and Linux

"lightning fast" Redmond Magazine

"covers all data sources" eWeek

"results in less than a second" InfoWorld

hundreds more reviews and developer case studies at www.dtsearch.com

dtSearch products:

Desktop with Spider	Web with Spider
Network with Spider	Engine for Win & .NET
Publish (portable media)	Engine for Linux
Document filters also available for separate licensing	

Ask about fully-functional evaluations

The Smart Choice for Text Retrieval® since 1991

www.dtSearch.com 1-800-IT-FINDS

msdn

magazine

JANUARY 2014 VOLUME 29 NUMBER 1

MOHAMMAD AL-SABT Editorial Director/mmeditor@microsoft.com

KENT SHARKEY Site Manager

MICHAEL DESMOND Editor in Chief/mmeditor@microsoft.com

DAVID RAMEL Technical Editor

SHARON TERDEMAN Features Editor

WENDY HERNANDEZ Group Managing Editor

SCOTT SHULTZ Creative Director

JOSHUA GOULD Art Director

SENIOR CONTRIBUTING EDITOR Dr. James McCaffrey

CONTRIBUTING EDITORS Rachel Appel, Dino Esposito, Kenny Kerr, Julie Lerman, Ted Neward, Charles Petzold, David S. Platt, Bruno Terkaly, Ricardo Villalobos

Redmond Media Group

Henry Allain President, Redmond Media Group

Michele Imgrund Sr. Director of Marketing & Audience Engagement

Tracy Cook Director of Online Marketing

Irene Fincher Audience Development Manager

ADVERTISING SALES: 818-674-3416/dlbianca@1105media.com

Dan LaBianca Vice President, Group Publisher

Chris Kourtoglou Regional Sales Manager

Danna Vedder Regional Sales Manager/Microsoft Account Manager

David Seymour Director, Print & Online Production

Serena Barnes Production Coordinator/msdnadproduction@1105media.com

1105 MEDIA

Neal Vitale President & Chief Executive Officer

Richard Vitale Senior Vice President & Chief Financial Officer

Michael J. Valenti Executive Vice President

Christopher M. Coates Vice President, Finance & Administration

Erik A. Lindgren Vice President, Information Technology & Application Development

David F. Myers Vice President, Event Operations

Jeffrey S. Klein Chairman of the Board

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in U.S. funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: *MSDN Magazine*, PO. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, PO. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: PO. Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o *MSDN Magazine*, 4 Venture, Suite 150, Irvine, CA 92618.

Legal Disclaimer: The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

Corporate Address: 1105 Media, Inc., 9201 Oakdale Ave., Ste 101, Chatsworth, CA 91311, www.1105media.com

Media Kits: Direct your Media Kit requests to Matt Morollo, VP Publishing, 508-532-1418 (phone), 508-875-6622 (fax), mmorollo@1105media.com

Reprints: For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International, Phone: 212-221-9595, E-mail: 1105reprints@parsintl.com, www.magreprints.com/QuickQuote.asp

List Rental: This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jlong@meritdirect.com; Web: www.meritdirect.com/1105

All customer service inquiries should be sent to MSDNmag@1105service.com or call 847-763-9560.



Printed in the USA

WPF lives!



➔ **XCEED Business Suite for WPF**

The essential set of WPF controls for all your line-of-business solutions. Includes the industry-leading **Xceed DataGrid for WPF**.
A total of 85 tools!



WinJS Takes the Next Step

We recently surveyed *MSDN Magazine* readers and found that, with the exception of C#, more companies report working with JavaScript than any other programming language—more than Java, more than C/C++, more than Visual Basic. So it should come as no surprise that this month's issue focuses on JavaScript development for Windows Runtime.

In his lead feature, “Build More Efficient Windows Store Apps Using JavaScript: Error Handling,” Eric Schmidt dives into some fundamental concepts of the Windows Library for JavaScript (WinJS), including promises, asynchronous debugging, and error handling. As Schmidt notes in an interview, the tooling and capabilities of WinJS differ from JavaScript in that the environment is tailored for full-fledged app development, rather than for Web sites. And these differences are important to understand when ensuring that flaws in code and logic are properly handled.

Schmidt describes the JavaScript `window.onerror` event as “the catchall for error handling” for Web apps.

Many of the techniques Schmidt explores in this month's feature are present in the initial version of Windows 8 and WinJS, but Schmidt says many developers are unclear on how the techniques work and relate to each other. He wrote this month's article with the idea of explaining “how the entire package of error-handling works together from a top-down, holistic perspective.”

For instance, Schmidt describes the JavaScript `window.onerror` event as “the catchall for error handling” for Web apps. Windows Store apps built using WinJS, however, leverage both `window.onerror` and the `WinJS.Application.onerror` event handler, which catches additional events like platform-specific errors and promises that error out. Schmidt also urges developers to consider moving to

Visual Studio 2013, which delivers key tooling improvements, including the ability to track down errors within chains of asynchronous calls—an important capability given that WinJS apps frequently chain promises to each other.

“When one of those promises errors out, you either need to have an `onError` parameter for each promise in the chain, or you need to track down the error starting from the last promise in the chain,” explains Schmidt, who says both approaches are time consuming. “The Tasks pane, which has been extended to encompass JavaScript code in Visual Studio 2013, really helps developers track down those errors in their code,” he notes.

The latest version of WinJS and its attendant tooling deliver what Schmidt describes as “ready to go” templates and controls that help streamline development. The updates also deliver significant performance improvements and streamlined WinJS app development.

“We provided new tools in Visual Studio 2013 for measuring HTML UI responsiveness, JavaScript memory usage, JavaScript function timing and asynchronous debugging,” Schmidt says. “In the platform, we improved the performance of some of our existing controls—for example, `ListView`—with APIs like the `Dispose` pattern and the `Scheduler`, as well as provided new controls that are better suited for specific tasks—for example, the `Repeater` control.”

Speaking of performance improvements, next month Schmidt will publish a follow-on article focused specifically on performance issues. What advice does he have for developers struggling to ensure that their WinJS applications are performant?

“The No. 1 thing I'd recommend is to really use the platform. Any retained-mode system—whether it's HTML or XAML or PDF or you name it—has certain sets of capabilities that are natively implemented and highly optimized within it. In as much as your app code and script calls into and leverages those capabilities, the better performance you'll get,” Schmidt says. He adds that the latest version of WinJS pushes more of the features lower down into the implementation, where they can be optimized for best performance.

Are you working with WinJS? If so, I'd love to hear about your experience. E-mail me at mmeditor@microsoft.com.

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

© 2014 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, *MSDN*, and Microsoft logos are used by 1105 Media, Inc. under license from owner.

NEW HOSTING

MAXIMUM FLEXIBILITY FOR YOUR WEB PROJECTS

ALL INCLUSIVE

- Included Domains: .com, .net, .org, .biz, .info
- Unlimited Power: webspace, traffic, mail accounts, SQL databases
- Linux or Windows operating system

POWERFUL APPS

- Over 140 popular apps (Drupal™, WordPress, Joomla!™, Typo3, Magento® and many more...)
- App Expert Support to answer your questions

POWERFUL TOOLS

- Premium software, including Adobe® Dreamweaver® CS5.5, and NetObjects Fusion® 2013
- 1&1 Mobile Website Builder
- **NEW:** PHP 5.5, Perl, Python, Ruby

SUCCESSFUL MARKETING

- 1&1 Search Engine Optimization
- Listing in business directories
- 1&1 E-Mail Marketing Manager
- Facebook® Credits

STATE-OF-THE-ART TECHNOLOGY

- Maximum Availability (Geo-redundancy)
- 300 Gbit/s network connection
- 2 GB RAM guaranteed
- **NEW:** Maximum performance with 1&1 CDN powered by CloudFlare™
- **NEW:** SiteLock security scan included!

HOSTING PACKAGES FOR PROFESSIONALS

STARTING AT

\$0⁹⁹ per month*



Call **1 (877) 461-2631**



1and1.com

* Offer valid for a limited time only. The \$0.99/month price reflects a 12-month pre-payment option for the 1&1 Starter Hosting package. Regular price of \$2.99/month after 12 months. Some features listed are only available with package upgrade. (Visit www.1and1.com for full details.) Program and pricing specifications and availability subject to change without notice. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are the property of their respective owners. © 2014 1&1 Internet. All rights reserved.



Using Regular Expressions with Modern C++

“C++ is a language for developing and using elegant and efficient abstractions.”—Bjarne Stroustrup

This quote from the creator of C++ really sums up what I love about the language. I get to develop elegant solutions to my problems by combining the language features and programming styles that I deem most suitable to the task.

C++11 introduced a long list of features that are in themselves quite exciting, but if all you see is a list of isolated features, then you’re missing out. The combination of these features makes C++ into the powerhouse that many have grown to appreciate. I’m going to illustrate this point by showing you how to use regular expressions with modern C++. The C++11 standard introduced a powerful regular expression library, but if you use it in isolation—using a traditional C++ programming style—you might find it somewhat tiresome. Unfortunately, this is the way that most of the C++11 libraries tend to be introduced. However, there is some merit in such an approach. If you were looking for a concise example of using some new library, it would be rather overwhelming to be forced into comprehending a slew of new language features at the same time. Still, the combination of C++ language and library features really turns C++ into a productive programming language.

To keep the examples focused on C++ and not on regular expressions, I’m necessarily going to use very simplistic patterns. You might wonder why I’m using regular expressions for such trivial problems, but it helps avoid getting lost in the mechanics of expression processing. Here’s a simple example: I’d like to match strings of names where the names might be formatted “Kenny Kerr” or “Kerr, Kenny.” I need to identify the first name and family name and then print them out in some consistent manner. First up is the target string:

```
char const s[] = "Kerr, Kenny";
```

To keep things simple, I’ll stick to char strings and I’ll avoid using the standard library’s `basic_string` class except to illustrate the results of certain matches. There’s nothing wrong with `basic_string`, but I find that most of the work I do with regular expressions tends to be targeted at memory-mapped files. Copying the contents of these files into string objects would only serve to slow down my applications. The standard library’s regular expression support is indifferent and perfectly happy to process sequences of characters without concern for how they’re managed.

The next thing I’ll need is a match object:

```
auto m = cmatch {};
```

This is really a collection of matches. The `cmatch` is a `match_results` class template that’s been specialized for char strings. At this point, the match “collection” is empty:

```
ASSERT(m.empty());
```

I’ll also need a pair of strings to receive the results:

```
string name, family;
```

I can now call the `regex_match` function:

```
if (regex_match(s, m, regex { R"((\w+) (\w+))" }))  
{  
}
```

This function attempts to match the pattern against the entire character sequence. This is in contrast to the `regex_search` function that’s quite happy to search for a match at any point within the string. I’m just creating the `regex` object “inline” for brevity, but this is not without cost. If you were going to match this regular expression repeatedly, you might be better off creating the `regex` object once and then holding on to it for the life of your application. The preceding pattern matches the names using the “Kenny Kerr” format. Assuming it’s a match, I can just copy out the substrings:

```
name = m[1].str();  
family = m[2].str();
```

The subscript operator returns the specified `sub_match` object. An index of zero represents the match as a whole, while subsequent indexes pinpoint any groups identified in the regular expression. Neither the `match_results` nor the `sub_match` object will create or allocate a substring. Instead, they delineate the range of characters with a pointer or iterator to the beginning and end of the match or submatch, producing the usual half-open range favored by the standard library. In this case, I’m explicitly calling the `str` method on each `sub_match` to create a copy of each submatch as string objects.

That handles the first possible format. For the second I need another call to `regex_match` with the alternative pattern (technically, you could match both formats with a single expression, but that’s beside the point):

```
else if (regex_match(s, m, regex { R"((\w+), (\w+))" }))  
{  
    name = m[2].str();  
    family = m[1].str();  
}
```

This pattern matches the names using the “Kerr, Kenny” format. Notice that I’ve had to reverse the indices, as the first group represented in this regular expression identifies the family name while the second identifies the first name. That’s about it for the `regex_match` function. **Figure 1** provides the complete listing for reference.

I don’t know about you, but the code in **Figure 1** looks tedious to me. While the regular expression library is certainly powerful and flexible, it’s not particularly elegant. I need to know about `match_results` and `sub_match` objects. I need to remember how this “collection” is indexed and how to extract the results. I could avoid making the copies, but it quickly becomes onerous.

I’ve already used a number of new C++ language features that you may or may not have come across before, but nothing should be overly startling. Now I want to show how you can use variadic templates to really spice up your regular expression usage. Rather than diving right in with more language features, I’m going to start by showing you a simple abstraction to simplify text processing so I can keep this practical and elegant.

Empower Your Customers



Create & Edit PDFs in .Net - ActiveX - WinRT

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .Net and ActiveX/COM
- New WinRT Component enables publishing C#, C++CX or Javascript apps to Windows Store
- New Postscript/EPS to PDF conversion module



Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft certified PDF Converter printer driver
- Export PDF documents into other formats such as JPeg, PNG, XAML or HTML5



Advanced HTML to PDF & XAML

- Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- Easy Integration and deployment within developer's applications
- WebkitPDF is based on the Webkit Open Source library and Amyuni PDF Creator



High Performance PDF Printer For Desktops and Servers



- Our high-performance printer driver optimized for Web, Application and Print Servers. Print to PDF in a fraction of the time needed with other tools. WHQL tested for Windows 32 and 64-bit including Windows Server 2012 and Windows 8
- Standard PDF features included with a number of unique features. Interface with any .Net or ActiveX programming language
- Easy licensing and deployment to fit system administrator's requirements



AMYUNI

All development tools available at

www.amyuni.com

USA and Canada

Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe

UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

First, I'll define a simple type to represent a sequence of characters that aren't necessarily null-terminated. Here's the strip class:

```
struct strip
{
    char const * first;
    char const * last;

    strip(char const * const begin,
          char const * const end) :
        first { begin },
        last { end }
    {}

    strip() : strip { nullptr, nullptr } {}
};
```

There are undoubtedly numerous such classes that I might reuse, but I find it helps to avoid too many dependencies when producing simple abstractions.

The strip class doesn't do much, but I'll augment it with a set of nonmember functions. I'll start with a pair of functions to define the range generically:

```
auto begin(strip const & s) -> char const *
{
    return s.first;
}

auto end(strip const & s) -> char const *
{
    return s.last;
}
```

Although not strictly necessary to this example, I find this technique provides a worthy measure of consistency with the standard library's containers and algorithms. I'll get back to the begin and end functions in a moment. Up next is the make_strip helper function:

```
template <unsigned Count>
auto make_strip(char const (&text)[Count]) -> strip
{
    return strip { text, text + Count - 1 };
}
```

This function comes in handy when attempting to create a strip from a string literal. For example, I can initialize a strip as follows:

```
auto s = make_strip("Kerr, Kenny");
```

Next, it's often useful to determine the length or size of the strip:

```
auto size(strip const & s) -> unsigned
{
    return end(s) - begin(s);
}
```

Here you can see I'm simply reusing the begin and end functions to avoid a dependency on the strip's members. I could protect the members of the strip class. On the other hand, it's often helpful to

be able to manipulate them directly from within an algorithm. Still, if I don't need to take a hard dependency, I won't.

Obviously, it's simple enough to create a standard string from a strip:

```
auto to_string(strip const & s) -> string
{
    return string { begin(s), end(s) };
}
```

This might come in handy if some of the results outlive the original character sequences. That rounds out the basic strip handling. I can initialize a strip and determine its size—and thanks to the begin and end functions, I can use a range-for statement to iterate over its characters:

```
auto s = make_strip("Kenny Kerr");

for (auto c : s)
{
    printf("%c\n", c);
}
```

When I first wrote the strip class, I was hoping I could call its members "begin" and "end" instead of "first" and "last." The trouble is that the compiler, when confronted with a range-for statement, first attempts to find suitable members that may be called as functions. If the target range or sequence doesn't include any members called begin and end, then the compiler looks for a suitable pair in the enclosing scope. The trouble is that if the compiler finds members called begin and end but they aren't suitable, it won't attempt to look any further. This might seem shortsighted, but C++ has complex name-lookup rules, and anything else would make it even more confusing and inconsistent.

The strip class is a simple little construct, but it doesn't do much in itself. I'll now combine it with the regular expression library to produce an elegant abstraction. I want to hide the mechanics of the match object, the tedious part of expression processing. This is where variadic templates come in. The key to understanding variadic templates is realizing you can separate the first argument from the rest. This typically results in compile-time recursion. I can define a variadic template to unpack a match object into subsequent arguments:

```
template <typename... Args>
auto unpack(cmatch const & m,
            Args & ... args) -> void
{
    unpack(sizeof...(Args)>(m, args...);
}
```

The "typename..." indicates that Args is a template parameter pack. The corresponding "..." in the type of args indicates that args is a function parameter pack. The "sizeof..." expression determines the number of elements in the parameter pack. The final "..." following args tells the compiler to expand the parameter pack into its sequence of elements.

The type of each argument may be different, but in this case, each will be a non-const strip reference. I'm using a variadic template so an unknown number of arguments can be supported. So far the unpack function doesn't appear to be recursive. It forwards its arguments to another unpack function with an additional template argument:

```
template <unsigned Total, typename... Args>
auto unpack(cmatch const & m,
            strip & s,
            Args & ... args) -> void
{
    auto const & v = m[Total - sizeof...(Args)];
    s = { v.first, v.second };
    unpack<Total>(m, args...);
}
```

Figure 1 The regex_match Reference Example

```
char const s[] = "Kerr, Kenny";
auto m = cmatch {};
string name, family;

if (regex_match(s, m, regex { R"((\w+) (\w+))" }))
{
    name = m[1].str();
    family = m[2].str();
}
else if (regex_match(s, m, regex { R"((\w+), (\w+))" }))
{
    name = m[2].str();
    family = m[1].str();
}
else
{
    printf("No match\n");
}
```


However, this unpack function separates the first argument following the match object from the rest. That's compile-time recursion in action. Assuming the args parameter pack isn't empty, it calls itself with the rest of the arguments. Eventually the sequence of arguments becomes empty and a third unpack function is needed to deal with this conclusion:

```
template <unsigned>
auto unpack(cmatch const &) -> void {}
```

This function doesn't do anything. It merely acknowledges the fact that the parameter pack may be empty. The previous unpack functions hold the key to unpacking the match object. The first unpack function captured the original number of elements in the parameter pack. This is necessary because each recursive call will effectively produce a new parameter pack with a diminishing size. Notice how I'm subtracting the size of the parameter pack from the original total. Given this total or stable size, I can index into the match collection to retrieve the individual submatches and copy their respective bounds into the variadic arguments.

That takes care of unpacking the match object. Although not required, I still find it helpful to hide the match object itself if it isn't needed directly—for example, if it's only needed to access the match prefix and suffix. I'll wrap up the whole thing to provide a simpler match abstraction:

```
template <typename... Args>
auto match(strip const & s,
           regex const & r,
           Args & ... args) -> bool
{
    auto m = cmatch {};

    if (regex_match(begin(s), end(s), m, r))
    {
        unpack<sizeof...(Args)>(m, args...);
    }

    return !m.empty();
}
```

This function is also a variadic template but isn't in itself recursive. It merely forwards its arguments to the original unpack function for processing. It also takes care of providing a local match object and defining the search sequence in terms of strip's begin and end helper functions. An almost identical function can be written to accommodate `regex_search` instead of `regex_match`. I can now rewrite the example from **Figure 1** far more simply:

```
auto const s = make_strip("Kerr, Kenny");
strip name, family;

if (match(s, regex { R"((\w+) (\w+))" }, name, family) ||
    match(s, regex { R"((\w+), (\w+))" }, family, name))
{
    printf("Match!\n");
}
```

How about iteration? The unpack functions also come in handy for handling the match results of an iterative search. Imagine a string with the canonical “Hello world” in a variety of languages:

```
auto const s =
    make_strip("Hello world/Hola mundo/Hallo wereId/Ciao mondo");
```

I can match each one with the following regular expression:

```
auto const r = regex { R"((\w+) (\w+))" };
```

The regular expression library provides the `regex_iterator` to iterate through the matches, but using iterators directly can become tedious. One option is to write a `for_each` function that calls a predicate for each match:

```
template <typename F>
auto for_each(strip const & s,
              regex const & r,
              F callback) -> void
{
    for (auto i = cregex_iterator { begin(s), end(s), r };
         i != cregex_iterator {};
         ++i)
    {
        callback(*i);
    }
}
```

I could then call this function with a lambda expression to unpack each match:

```
for_each(s, r, [] (cmatch const & m)
{
    strip hello, world;
    unpack(m, hello, world);
});
```

This certainly works, but I always find it frustrating that I can't easily break out of this type of loop construct. The range-for statement provides a more convenient alternative. I'll begin by defining a simple iterator range that the compiler will recognize to implement the range-for loop:

```
template <typename T>
struct iterator_range
{
    T first, last;

    auto begin() const -> T { return first; }
    auto end() const -> T { return last; }
};
```

I can now write a simpler `for_each` function that just returns an `iterator_range`:

```
auto for_each(strip const & s,
              regex const & r) -> iterator_range<cregex_iterator>
{
    return
    {
        cregex_iterator { begin(s), end(s), r },
        cregex_iterator {}
    };
}
```

The compiler will take care of producing the iteration and I can simply write a range-for statement with minimal syntactic overhead, breaking early if I so choose:

```
for (auto const & m : for_each(s, r))
{
    strip hello, world;
    unpack(m, hello, world);

    printf("%.*s' '%.*s'\n",
           size(hello), begin(hello),
           size(world), begin(world));
}
```

The console presents the expected results:

```
'Hello' 'world'
'Hola' 'mundo'
'Hallo' 'wereId'
'Ciao' 'mondo'
```

C++11 and beyond provide an opportunity to revitalize C++ software development with a modern style of programming that lets you produce elegant and efficient abstractions. The regular expression grammar can humble even the most seasoned of developers. Why not spend a few minutes developing a more elegant abstraction? At least the C++ portion of your task will be a pleasure! ■

KENNY KERR is a computer programmer based in Canada, an author for *Pluralsight* and a Microsoft MVP. He blogs at kennykerr.ca and you can follow him on Twitter at twitter.com/kennykerr.

WORKING WITH FILES?



- ✓ CONVERT
- ✓ PRINT
- ✓ CREATE
- ✓ COMBINE
- ✓ MODIFY

100% Standalone - No Office Automation

US Sales:
+1 888 277 6734
sales@aspose.com

European Sales:
+44 141 416 1112
sales.europe@aspose.com



SCAN FOR
20% SAVINGS



ASPOSE.TOTAL



Every Aspose component combined in
ONE powerful suite!

Powerful File Format Components and Controls

- ▶ **Aspose.Words**
DOC, DOCX, RTF, HTML, PDF, XPS & other document formats.
 - ▶ **Aspose.Cells**
XLS, XLSX, XLSM, XLTX, CSV, SpreadsheetML & image formats.
 - ▶ **Aspose.BarCode**
JPG, PNG, BMP, GIF, TIF, WMF, ICON & other image formats.
 - ▶ **Aspose.Pdf**
PDF, XML, XLS-FO, HTML, BMP, JPG, PNG & other image formats.
 - ▶ **Aspose.Email**
MSG, EML, PST, EMLX & other formats.
 - ▶ **Aspose.Slides**
PPT, PPTX, POT, POTX, XPS, HTML, PNG, PDF & other formats.
 - ▶ **Aspose.Diagram**
VSD, VSDX, VSS, VST, VSX & other formats.
- ... and many others!*

Aspose.Total for .NET
Aspose.Total for Java

Aspose.Total for Cloud
Aspose.Total for Android

Get your FREE evaluation copy at www.aspose.com

.NET

Java

Cloud

Android



Code First Goodies in Entity Framework 6

In my December 2013 article, “Entity Framework 6: The Ninja Edition” (msdn.microsoft.com/magazine/dn532202), I described many of the new features in Entity Framework 6 (EF6). I wasn’t able to delve into every feature, though, so this month I’ll drill into some of the EF6 enhancements specific to Code First. Two of the features I’ll discuss are relevant to Code First mappings, and the others relate to Code First migrations.

Load Many Fluent API Mappings at Once

There are two ways to specify Fluent mappings (from your classes to your database) for a model. One is directly in the `OnModelCreating` method of the `DbContext` class, like this:

```
modelBuilder.Entity<Casino>()
    .Property(c => c.Name).IsRequired().HasMaxLength(200);
modelBuilder.Entity<PokerTable>()
    .Property(c => c.SerialNo).HasColumnName("SerialNumber");
```

When you have a lot of mappings, you can organize them by type into individual `EntityTypeConfiguration` classes, which you then add into the model builder using code like this:

```
modelBuilder.Configurations.Add(new CasinoConfiguration());
modelBuilder.Configurations.Add(new PokerTableConfiguration());
```

However, if you have a lot of mappings for a lot of entities, you can end up with many repetitive `modelBuilder.Configurations.Add` methods in `OnModelCreating`. To eliminate this drudgery, you can now load all `EntityTypeConfiguration` classes from a given assembly with a single method. Here I use the new `AddFromAssembly` method to load configurations that are specified in the executing assembly for the running application:

```
modelBuilder.Configurations
    .AddFromAssembly(Assembly.GetExecutingAssembly());
```

A nice feature of this method is that it isn’t restricted by the scope of the configurations it will load. The custom `EntityTypeConfiguration` classes can even be marked private and the method will find them. Moreover, `AddFromAssembly` also comprehends inheritance hierarchies in `EntityTypeConfiguration` classes.

`AddFromAssembly` is one of a number of community contributions from Unai Zorrilla. See Zorrilla’s blog post, “EF6: Setting Configurations Automatically,” at bit.ly/160BuJ5 for more details—and leave him a note of thanks while you’re there.

Define Your Own Default Schema

In my March 2013 Data Points column, “Playing with the EF6 Alpha” (msdn.microsoft.com/magazine/jj991973), I talked a bit about schema

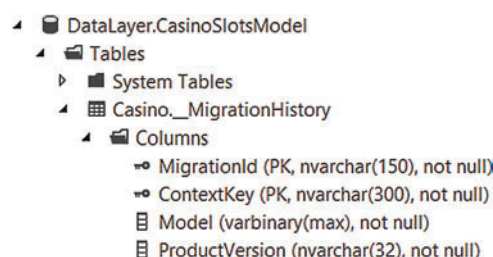


Figure 1 Default Schema for the `__MigrationHistory` Table

support for Code First. One new feature is a mapping you can configure in `OnModelCreating`: `HasDefaultSchema`. This lets you specify the database schema for all of the tables a context maps to, rather than using the EF default of `dbo`. In the “Entity Framework 6: The Ninja Edition” article, I executed some raw SQL in the discussion about `DbTransactions`:

```
("Update Casino.Casinos set rating= " + (int) Casino.Rating)
```

You may have noticed the `Casino` schema I specified in the SQL. The reason I have a schema named `Casino` is because I specified it in the `OnModelCreating` method of my `DbContext` (`CasinoSlotsModel`):

```
modelBuilder.HasDefaultSchema("Casino");
```

I also talked about the new support in EF6 for Code First migrations that are being run against a database that has different schemas in it. Because that hasn’t changed since EF6 Alpha, I’ll let you read about it in my earlier column.

Migrations Scripts to Rebuild the Database from Any Point

One of the new features for Code First migrations listed in the specs (and in every article that simply reiterates the specs) is “idempotent migrations scripts.” Now, you may be someone with a comp sci degree under your belt, or perhaps you’re a DBA. But I am neither and I had to look up the meaning of “idempotent.” According to Wikipedia, which references an IBM engineer (bit.ly/9MlrRK): “In computer science, the term ‘idempotent’ is used ... to describe an operation that will produce the same results if executed once or multiple times.” I also had to look up how to pronounce it. It’s *eye-dem-poe-tent*.

In the database world, idempotent can be used to describe a SQL script that always has the same impact on a database regardless of its state. With Code First migrations, before running a migration, such a script will check to see if that migration has already been run. This feature is specific to the `-script` parameter of `Update-Database`.

Code download available at msdn.microsoft.com/magazine/msdnmag0114.

Windows. Web. Mobile.

Your next great app starts here.

DevExpress .NET controls, frameworks and libraries were built with you in mind. Built for those who demand the highest quality and expect the best performance... for those who require reliable tools engineered to meet today's needs and address tomorrow's requirements.

Experience the DevExpress Difference today and download your free 30-day trial and let's build great apps, together.



WinForms



ASP.NET



WPF



Silverlight



Windows 8 XAML



HTML JS



Reporting



DevExtreme Mobile



Learn more and download your free trial
devexpress.com/try



All trademarks or registered trademarks are property of their respective owners.

Figure 2 Custom HistoryContext to Redefine __MigrationHistory Table

```
public class CustomHistoryContext : HistoryContext
{
    public CustomHistoryContext
        (DbConnection dbConnection, string defaultSchema)
        : base(dbConnection, defaultSchema)
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.Entity<HistoryRow>()
            .ToTable("__MigrationHistory", "admin");
    }
}
```

EF has always provided the ability to create scripts that run through all of the migration steps from a particular starting point (source) and optionally to an explicit endpoint (target). This NuGet command evokes that behavior:

```
Update-Database -Script
-SourceMigration:NameOfStartMigration
-TargetMigration:NameOfEndMigrationThatIsntLatest
```

What's new is that the generated script is much smarter now when you call the command in a particular way:

```
Update-Database -Script -SourceMigration $InitialDatabase
```

With this particular command, EF adds logic to the script that checks to see which migrations have already been applied before executing the SQL for a particular migration.

This also happens to work if you replace \$InitialDatabase with 0, though there's no guarantee this alternative will be supported in future versions.

In response, the script starts with the initial migration and carries through to the latest migration. This is why the syntax does not explicitly supply the names of the target or source migrations.

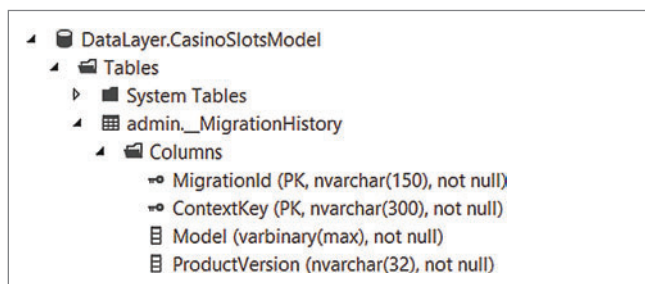


Figure 3 Customized __MigrationHistory Table

Figure 4 DbMigrations.CreateTable Method

```
CreateTable(
    Casino.SlotMachines",
    c => new
    {
        Id = c.Int(nullable: false, identity: true),
        SlotMachineType = c.Int(nullable: false),
        SerialNumber = c.String(maxLength: 4000),
        HotelId = c.Int(nullable: false),
        DateInService = c.DateTime(nullable: false),
        HasQuietMode = c.Boolean(nullable: false),
        LastMaintenance = c.DateTime(nullable: false),
        Casino_Id = c.Int(),
    })
    .PrimaryKey(t => t.Id)
    .ForeignKey("Casino.Casinos", t => t.Casino_Id)
    .Index(t => t.Casino_Id);
```

But with this particular command, EF6 adds logic to the script that checks to see which migrations have already been applied before executing the SQL for a particular migration. Here's an example of code you'll see in the script:

```
IF @CurrentMigration < '201310311821192_AddedSomeNewPropertyToCasino'
BEGIN
    ALTER TABLE [Casino].[Casinos] ADD [AgainSomeNewProperty] [nvarchar](4000)
    INSERT [Casino].[__MigrationHistory]([MigrationId], [ContextKey],
    [Model], [ProductVersion])
    VALUES (N'201310311821192_AddedSomeNewPropertyToCasino', N'CasinoModel.
    Migrations.Configuration', HugeBinaryValue , N'6.1.0-alpha1-21011')
END
```

The code checks in the __MigrationHistory table to see if the AddedSomeNewPropertyToCasino script has been run on the database yet. If it has, the SQL from that migration won't be executed. Prior to EF6, the script would just run the SQL without checking to see if it had already been run.

Provider-Friendly Migration History Tables

EF6 lets you customize how the __MigrationHistory table is defined using a feature called the Customizable Migrations History Table. This is important if you're using third-party data providers that have different requirements than the defaults. Figure 1 shows the default schema of the table.

Here's an example of how this can be useful. On CodePlex, a developer noted that because each char in the two PKs can be greater than 1 byte, he was getting an error that the length of the compound key created from MigrationId and ContextKey exceeded the allowed key length for a MySQL table: 767 bytes (bit.ly/18rw1BX). To resolve the issue, the MySQL team is using the HistoryContext internally to alter the lengths of the two key columns while working on the EF6 version of MySQL Connector for ADO.NET (bit.ly/7uYw2a).

Notice in Figure 1 that the __MigrationHistory table gets the same schema I defined for the context

```
AddColumn
AddForeignKey
AddPrimaryKey
AlterColumn
AlterStoredProcedure
CreateIndex
CreateStoredProcedure
CreateTable<>
DropColumn
DropForeignKey
DropIndex
DropPrimaryKey
DropStoredProcedure
DropTable
MoveStoredProcedure
MoveTable
RenameColumn
RenameStoredProcedure
RenameTable
Sql
```

Figure 5 DbMigrations Database Schema Operations

using HasSchema: Casino. You may have conventions that say non-data tables should be used by a schema with limited permissions, so you might want to change the schema name of that table. You can do that with the HistoryContext, for example to specify that the “admin” schema be used.

HistoryContext derives from DbContext, so the code should be somewhat familiar if you’ve worked with DbContext and Code First in the past. **Figure 2** shows a HistoryContext class I defined to specify the admin schema.

You can also use familiar API calls like Property().HasColumnType, HasMaxLength or HasColumnName. For example, if you need to change the length of the ContextKey, you can do this:

```
modelBuilder.Entity<HistoryRow>()
    .Property(h => h.ContextKey).HasMaxLength(255);
```

HistoryContext
is a powerful feature
but it needs to be
used carefully.

If you’ve already read last month’s article, you should be familiar with EF6 DbConfiguration. This is what you use to let your model know about the CustomHistoryContext file. In your custom DbConfiguration’s constructor, you need to specify the HistoryContext to be used. Here I set the context for the SQL Server provider to use CustomHistoryContext:

```
SetHistoryContext(
    SqlProviderServices.ProviderInvariantName,
    (connection, defaultSchema) =>
        new CustomHistoryContext(connection,
            defaultSchema));
```

Database initialization and migrations functionality will see this additional context and construct SQL accordingly. The table in **Figure 3** was created using the custom HistoryContext to change the schema name of the __MigrationHistory table to admin. (I didn’t include the sample code for changing the column length.)

HistoryContext is a powerful feature but it needs to be used carefully. Hopefully the database provider you’re using will have already used it to specify a __MigrationHistory table that’s relevant to the target database, and you won’t even need to think about this. Otherwise, I recommend checking the MSDN document on this feature and heeding its guidance (bit.ly/16eK2pD).

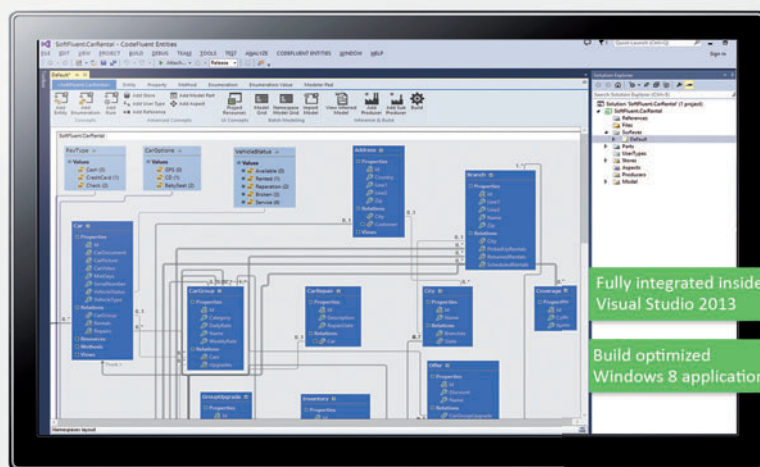
Create Custom Migration Operations

If you’ve used migrations before—not automatically but by explicitly creating and executing migrations from the Package Manager Console window—you may have explored the migration files created by add-migration. If so, you might have discovered that Code First migrations have a strongly typed API to describe each change to make to the database schema: System.Data.Entity.Migrations.DbMigration.

Figure 4 shows an example of the CreateTable method that sets a number of attributes.

Save your time!

Stop writing repetitive code and focus on what matters



Fully integrated inside
Visual Studio 2013

Build optimized
Windows 8 applications

Generate rock-solid foundations for your .NET applications



“ Benefit from out-of-the-box advanced features

A remarkable product that adds features where all the ‘junior’ equivalents fall short. Things like hassle-free schema updates, up-casting, enum & null management, security, full data-binding including grids with pagination, performance, interface support, custom stored procedures within a wide range of architectures.

Basically all the ‘add-ons’ you discover you need when you start developing a real world app based on any code generator.

”

Boris Bosnjak, Developer, Dreamquest, Canada

GET A LICENCE
WORTH \$399
FOR FREE

Go to www.softfluent.com/forms/msdn-2014

Tools for developers, by developers.
More information at www.softfluent.com
Contact us at info@softfluent.com

SoftFluent



Figure 6 A Custom Migration Operation for Creating a Database View

```
public class CreateViewOperation : MigrationOperation
{
    public CreateViewOperation(string viewName, string viewQueryString)
        : base(null)
    {
        ViewName = viewName;
        ViewString = viewQueryString;
    }

    public string ViewName { get; private set; }
    public string ViewString { get; private set; }

    public override bool IsDestructiveChange
    {
        get { return false; }
    }
}
```

Figure 7 Custom SqlServerMigrationSqlGenerator Class

```
public class CustomSqlServerMigrationSqlGenerator
    : SqlServerMigrationSqlGenerator
{
    protected override void Generate(MigrationOperation migrationOperation)
    {
        var operation = migrationOperation as CreateViewOperation;

        if (operation != null)
        {
            using (IndentedTextWriter writer = Writer())
            {
                writer.WriteLine("CREATE VIEW {0} AS {1} ; ",
                                operation.ViewName,
                                operation.ViewString);
                Statement(writer);
            }
        }
    }
}
```

Figure 8 Using the New CreateView Operation

```
public partial class AddView : DbMigration
{
    public override void Up()
    {
        this.CreateView("dbo.CasinosWithOver100SlotMachines",
            @"SELECT *
            FROM Casino.Casinos
            WHERE Id IN (SELECT CasinoId AS Id
            FROM Casino.SlotMachines
            GROUP BY CasinoId
            HAVING COUNT(CasinoId)>=100)");
    }

    public override void Down()
    {
        this.RemoveView("dbo.CasinosWithOver100SlotMachines");
    }
}
```

Providers then translate those API calls into database-specific SQL.

There are methods to create tables and indexes, create or alter properties, drop objects and more. It's a fairly rich API, as you can see from the possibilities listed in **Figure 5**—which include the ability to simply execute some SQL directly. But, in some cases, it may not be rich enough for your needs. For example, there's no method for creating database views, specifying permissions or plenty other operations.

Once again, the community came to the rescue. In EF6, you now have the ability to create custom migrations operations you can call by customizing the migration classes generated

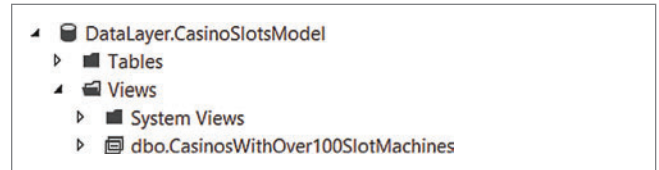


Figure 9 The Newly Created View Generated by Update-Database

by add-migration. This is thanks to another developer in the CodePlex community, Iñaki Elcoro, aka iceclow.

To create your own operation, you have to perform a few steps. I'll show the core of each step. You can see the full code and how the steps are organized in the download for this article.

1. Define the operation. Here I've defined a CreateView-Operation, as listed in **Figure 6**.
2. Create an extension method to point to the operation. This makes it simple to call from DbMigration:
3. Define the SQL for the operation in the Generate method of a custom SqlServerMigrationSqlGenerator class, as shown in **Figure 7**.
4. Tell the DbConfiguration class to use the custom SqlServerMigrationSqlGenerator class:

```
SetMigrationSqlGenerator("System.Data.SqlClient",
    () => new CustomSqlServerMigrationSqlGenerator());
```

With all of this in place, you can now use the new operation in a migration file and Update-Database will know what to do with it. **Figure 8** shows the CreateView operation in use, and provides a reminder that you'll also need to create an operation to remove the view, which would be called by the Down method if you need to unwind this migration.

After I call Update-Database, you can see the new view in my database in **Figure 9**.

Code First Continues to Evolve

With the core features of Code First in place, Microsoft and developers from the community took the opportunity to start putting some polish on EF6, with features that now benefit from more flexibility. But it doesn't stop with the release of EF6. If you filter the CodePlex work items for versions beyond 6.0.1 (at bit.ly/1dA0LZf), you can see that even more polish is being added to future releases of EF6 and Code First. These items are in various states. Perhaps you'll find one on which you'd like to work. ■

JULIE LERMAN is a Microsoft MVP .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other Microsoft .NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (2010) as well as a Code First edition (2011) and a DbContext edition (2012), all from O'Reilly Media. Follow her on Twitter at twitter.com/julielerman and see her Pluralsight courses at julieme.com/PS-Videos.

THANKS to the following technical expert for reviewing this article:
Rowan Miller (Microsoft)



The world's leading Imaging SDK
RUNS ANYWHERE



Document

OCR, Barcode & Forms Recognition

PDF Read, Write & Edit

Cleanup and Preprocessing



Medical

DICOM

PACS

Medical Workstation



Multimedia

Playback, Capture & Conversion

MPEG-2 Transport Stream

DVR



Imaging

Viewers

Image processing

150+ Formats

C++

C#

JavaScript

VB

Objective-C

Java

.NET

Windows API

WinRT

Linux

iOS

OS X

Android

HTML5





EXPERT SOLUTIONS FOR .NET DEVELOPERS

The figure consists of four 10x10 grids, each containing blue dots. The first grid shows a solid 10x10 square. The second grid shows a 10x10 square with a 4x4 square removed from its center. The third grid shows a 10x10 square with a 6x6 square removed from its center. The fourth grid shows a 10x10 square with a 10x4 vertical bar removed from its center, leaving a cross-like shape.

[illegible]

Microsoft



Visual Studio

msdn

Visual Studio



1105 MEDIAZ

LAS VEGAS 2014

March 10 - 14 | Planet Hollywood Hotel & Casino



Visual Studio LIVE! YOUR GUIDE TO THE .NET DEVELOPMENT UNIVERSE

LIVE!

Visual Studio Live! is your guide to the .NET Development universe, featuring code-filled days, networking nights and independent education. Whether you are a .NET developer, software architect or a designer, Visual Studio Live!'s multi-track events include focused, cutting-edge education on the .NET platform that you'll be ready to implement as soon as you get back to the office.

COMPREHENSIVE TRAINING FOR THE DEVELOPER WORLD.

Visual Studio Live! Las Vegas is part of Live! 360 DEV, which means you'll have access to four (4) other co-located events at **no additional cost:**

SQL Server LIVE!
SQL SERVER FOR MODERN DEVELOPERS

SharePoint LIVE!
TRAINING FOR COLLABORATION

ModernApps LIVE!
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

AND INTRODUCING Web Dev Live!

Web Dev LIVE!
HTML5, JAVASCRIPT & ASP.NET TRAINING

Five (5) events means over a hundred sessions to choose from – mix and match sessions to create your own, custom event line-up - it's like no other dev conference available today!

**Register by January 22
and Save \$400!**

Use promo code VSLJAN4

vslive.com/lasvegas



Scan the QR code to register or for more event details.

TURN THE PAGE FOR MORE EVENT DETAILS →



VISUAL STUDIO LIVE! TRACKS

Visual Studio / .NET Framework	Windows 8.1 / WinRT	WPF / Silverlight	Cloud Computing	Windows Phone	Cross-Platform Mobile Development
--------------------------------	---------------------	-------------------	-----------------	---------------	-----------------------------------

WEB DEV LIVE! TRACKS

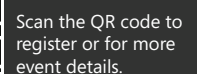
ASP.NET	HTML5
---------	-------

VISUAL STUDIO LIVE!

WEB DEV LIVE!

START TIME	END TIME	Pre-Conference Workshops: Monday, March 10, 2014 (Separate entry fee required)				
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries				
9:00 AM	6:00 PM	LWM01 Workshop: Building Distributed Modern Apps in C# and XAML - <i>Rockford Lhotka</i>		LWM02 Workshop: Modern UX Design - <i>Billy Hollis</i>		LWM03 Workshop: Data-Centric Single Page Applications with Knockout, Durandal, Breeze, and Web API - <i>Brian Noyes</i>
7:00 PM	9:00 PM	Dine-A-Round Dinner				
START TIME	END TIME	Day 1: Tuesday, March 11, 2014				
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries				
8:00 AM	9:00 AM	Keynote: To Be Announced				
9:15 AM	10:30 AM	LT01 Visual Studio 2013 New IDE Features Part I - <i>Deborah Kurata</i>	LT02 What's New for HTML/WinJS Windows Store Apps - <i>Ben Dewey</i>	LT03 Introduction to Azure - <i>Vishwas Lele</i>	LT04 HTML5 for Better Web Sites - <i>Robert Boedigheimer</i>	
10:45 AM	12:00 PM	LT08 Visual Studio 2013 New IDE Features Part II - <i>Deborah Kurata</i>	LT09 What's New for XAML Windows Store Apps - <i>Ben Dewey</i>	LT10 Windows Azure Cloud Services - <i>Vishwas Lele</i>	LT11 12 Things Every Developer Really Needs to Know About JavaScript - <i>Rachel Appel</i>	
12:00 PM	1:15 PM	Lunch • Visit the EXPO				
1:15 PM	2:15 PM	LT15 - Chalk Talk: To Be Announced				LT16 Chalk Talk: Introduction to Speech Recognition with C# - <i>James McCaffrey</i>
2:30 PM	3:45 PM	LT19 What's New in Team Foundation Server 2013 - <i>Benjamin Day</i>	LT20 Controlling Hardware Using Windows 8.1 - <i>Brian Peek</i>	LT21 WCF & Web API: Can We All Just Get Along?!? - <i>Miguel Castro</i>	LT22 Great User Experiences with CSS 3 - <i>Robert Boedigheimer</i>	
3:45 PM	4:15 PM	Networking Break • Visit the EXPO				
4:15 PM	5:30 PM	LT26 Visual Studio Online: Cloud-y Visual Studio & TFS for Your Teams - <i>Benjamin Day</i>	LT27 Performance and Diagnostics Hub in Visual Studio 2013 - <i>Brian Peek</i>	LT28 Windows Azure Web Sites - <i>Vishwas Lele</i>	LT29 Writing Next Generation Javascript with TypeScript - <i>Rachel Appel</i>	
5:30 PM	7:00 PM	Welcome Reception				
START TIME	END TIME	Day 2: Wednesday, March 12, 2014				
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries				
8:00 AM	9:00 AM	Keynote: To Be Announced				
9:15 AM	10:30 AM	LW01 Getting Started with Windows Phone Development - <i>Nick Landry</i>	LW02 Test Your XAML-based Windows Store Apps with VS2013 - <i>Benjamin Day</i>	LW03 Overview of IaaS in Windows Azure with Virtual Machines - <i>Eric D. Boyd</i>	LW04 Using jQuery to Replace the Ajax Control Toolkit - <i>Robert Boedigheimer</i>	
10:45 AM	12:00 PM	LW08 Developing Windows Phone Apps with Azure - <i>Rockford Lhotka</i>	LW09 Accelerate Your App Design with Blend for Visual Studio 2013 - <i>Brian Noyes</i>	LW10 Building Real-time, Multi-user Interactive Web and Mobile Applications Using SignalR - <i>Marcel de Vries</i>	LW11 Building Web Applications Using Kendo UI and the MVVM Pattern - <i>Ben Hoelting</i>	
12:00 PM	1:15 PM	Birds-of-a-Feather Lunch • Visit the EXPO				
1:15 PM	2:30 PM	LW15 Cross Platform Native Mobile App Development for iOS, Android and Windows Using the Power of C# - <i>Marcel de Vries</i>	LW16 Holy Updates! (Or What's New in Windows 8.1) - <i>Philip Japikse</i>	LW17 Persistence In The Cloud: How to Use Azure Storage - <i>David Giard</i>	LW18 Angular for .NET Developers - <i>Jesse Liberty</i>	
2:45 PM	4:00 PM	LW22 Mobile App Development for the Web Developer Using PhoneGap - <i>Eric D. Boyd</i>	LW23 WinRT Business Application Architecture - <i>Rockford Lhotka</i>	LW24 Building Services with ASP.NET MVC Web API Deep Dive - <i>Marcel de Vries</i>	LW25 Build Your First Angular Web Application - <i>Matthew DiFranco</i>	
4:00 PM	4:30 PM	Sponsored Break • Visit the EXPO • Expo Raffle @ 4:15PM				
4:30 PM	5:45 PM	LW29 iOS Development Survival Guide for the .NET Guy - <i>Nick Landry</i>	LW30 Overview of the Bing Platform - <i>Brian Peek</i>	LW31 Debugging and Monitoring Windows Azure Cloud Services - <i>Eric D. Boyd</i>	LW32 Connecting the Dots: Using HTML5, jQuery, and Web API Together - <i>David Giard</i>	
7:00 PM	9:00 PM	Live! 360 Evening Event				
START TIME	END TIME	Day 3: Thursday, March 13, 2014				
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries				
8:00 AM	9:15 AM	LTH01 What's New in .NET 4.5.1 - <i>Jason Bock</i>	LTH02 Interaction Design Principles and Patterns - <i>Billy Hollis</i>	LTH03 To Be Announced		LTH04 Knocking it Out of the Park with Knockout.JS - <i>Miguel Castro</i>
9:30 AM	10:45 AM	LTH08 Building Shared Mobile Apps for Windows Store and Windows Phone - <i>Nick Landry</i>	LTH09 Applying UX Design in XAML - <i>Billy Hollis</i>	LTH10 Learning Entity Framework 6 - <i>Leonard Label</i>	LTH11 Structuring Your Single Page Application with Durandal - <i>Brian Noyes</i>	
11:00 AM	12:15 PM	LTH15 Visual Studio Online—It's More Than You Know - <i>Brian Randell</i>	LTH16 Creating Games for Windows 8 with Unity - <i>Brian Lagunas</i>	LTH17 To Be Announced		LTH18 Create an API For Your Application With Service Stack - <i>Jesse Liberty</i>
12:15 PM	1:30 PM	Lunch				
1:30 PM	2:45 PM	LTH22 App Insights - App Performance Monitoring - <i>Brian Randell</i>	LTH23 Building Composite XAML Applications with Prism - <i>Brian Lagunas</i>	LTH24 Asynchronous Debugging in .NET - <i>Jason Bock</i>	LTH25 Busy Developer's Guide to Node.js - <i>Ted Neward</i>	
3:00 PM	4:15 PM	LTH29 App Insights - Global System Monitoring - <i>Brian Randell</i>	LTH30 Implementing MVVM (Model-View-View Model) for WPF - <i>Philip Japikse</i>	LTH31 Browser as Code Editor: A Tour of Monaco - <i>Jason Bock</i>	LTH32 Busy Developer's Guide to Everything Not-JavaScript - <i>Ted Neward</i>	
START TIME	END TIME	POST-CONFERENCE WORKSHOPS: FRIDAY, MARCH 14, 2014 (SEPARATE ENTRY FEE REQUIRED)				
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries				
8:00 AM	05:00 PM	LWF01 Workshop: Deep Dive into Visual Studio 2013, TFS, and Visual Studio Online - <i>Brian Randell</i>		LWF02 Workshop: Build and Submit a Game to the Windows App Store - <i>David Giard</i>		LWF03 Workshop: Busy Developer's Guide to AngularJS - <i>Ted Neward</i>

Speakers and sessions subject to change



Build More Efficient Windows Store Apps Using JavaScript: Error Handling

Eric Schmidt

Believe it or not, sometimes app developers write code that doesn't work. Or the code works but is terribly inefficient and hogs memory. Worse yet, inefficient code results in a poor UX, driving users crazy and compelling them to uninstall the app and leave bad reviews.

I'm going to explore common performance and efficiency problems you might encounter while building Windows Store apps with JavaScript. In this article, I take a look at best practices for error handling using the Windows Library for JavaScript (WinJS). In a future article, I'll discuss techniques for doing work without

blocking the UI thread, specifically using Web Workers or the new WinJS.Utilities.Scheduler API in WinJS 2.0, as found in Windows 8.1. I'll also present the new predictable-object lifecycle model in WinJS 2.0, focusing particularly on when and how to dispose of controls.

For each subject area, I focus on three things:

- Errors or inefficiencies that might arise in a Windows Store app built using JavaScript.
- Diagnostic tools for finding those errors and inefficiencies.
- WinJS APIs, features and best practices that can ameliorate specific problems.

I provide some purposefully buggy code but, rest assured, I indicate in the code that something is or isn't supposed to work.

I use Visual Studio 2013, Windows 8.1 and WinJS 2.0 for these demonstrations. Many of the diagnostic tools I use are provided in Visual Studio 2013. If you haven't downloaded the most-recent versions of the tools, you can get them from the Windows Dev Center (bit.ly/K8nkk1). New diagnostic tools are released through Visual Studio updates, so be sure to check for updates periodically.

I assume significant familiarity with building Windows Store apps using JavaScript. If you're relatively new to the platform, I suggest beginning with the basic "Hello World" example (bit.ly/vbVHC) or, for more of a challenge, the Hilo sample for JavaScript (bit.ly/SgIOAA).

Setting up the Example

First, I create a new project in Visual Studio 2013 using the Navigation App template, which provides a good starting point for a basic multipage app. I also add a NavBar control (bit.ly/14vfvih) to the default.html page at the root of the solution, replacing the

GET HELP BUILDING YOUR WINDOWS STORE APP!

Receive the tools, help and support you need to develop your Windows Store apps.

bit.ly/XLjOrx

This article discusses:

- Creating the sample project in Visual Studio 2013 using the Navigation App template
- Application-, page- and navigation-level error handling
- Asynchronous JavaScript debugging

Technologies discussed:

Windows 8.1, Windows Library for JavaScript 2.0, Visual Studio 2013

Code download will be posted when available. Please check the [online version of the article](#).

Figure 1 The NavBar Control

```
<!-- The global navigation bar for the app. -->
<div id="navBar" data-win-control="WinJS.UI.NavBar">
  <div id="navContainer" data-win-control="WinJS.UI.NavBarContainer">
    <div id="homeNav" data-win-control="WinJS.UI.NavBarCommand"
      data-win-options="{
        location: '/pages/home/home.html',
        icon: 'home',
        label: 'Home page'
      }">
    </div>
    <div id="handlingErrors"
      data-win-control="WinJS.UI.NavBarCommand"
      data-win-options="{
        location: '/pages/handlingErrors/handlingErrors.html',
        icon: 'help',
        label: 'Handling errors'
      }">
    </div>
    <div id="chainedAsync"
      data-win-control="WinJS.UI.NavBarCommand"
      data-win-options="{
        location: '/pages/chainedAsync/chainedAsync.html',
        icon: 'link',
        label: 'Chained asynchronous calls'
      }">
    </div>
  </div>
</div>
```

AppBar code the template provided. Because I want to demonstrate multiple concepts, diagnostic tools and programming techniques, I'll add a new page to the app for each demonstration. This makes it much easier for me to navigate between all the test cases.

The complete HTML markup for the NavBar is shown in **Figure 1**. Copy and paste this code into your solution if you're following along with the example.

For more information about building a navigation bar, check out some of the Modern Apps columns by Rachel Appel, such as the one at msdn.microsoft.com/magazine/dn342878.

You can run this project with just the navigation bar, except that clicking any of the navigation buttons will raise an exception in `navigator.js`. Later in this article, I'll discuss how to handle errors that come up in `navigator.js`. For now, remember the app always starts on the home-page and you need to right-click the app to bring up the navigation bar.

Handling Errors

Obviously, the best way to avoid errors is to release apps that don't raise errors. In a perfect world, every developer would write perfect code that never crashes and never raises an exception. That perfect world doesn't exist.

As much as users prefer apps that are completely error-free, they are exceptionally good at finding new and creative ways to break apps—ways you never dreamed of. As a result, you need to incorporate robust error handling into your apps.

Errors in Windows Store apps built with JavaScript and HTML act just like errors in normal Web pages. When an error happens in a Document Object Model (DOM) object that allows for error handling (for example, the `<script>`, `<style>` or `` elements), the `onerror` event for that element is raised. For errors in the JavaScript call stack, the error travels up the chain of calls until caught (in a try/catch block, for instance) or until it reaches the window object, raising the `window.onerror` event.

WinJS provides several layers of error-handling opportunities for your code in addition to what's already provided to normal Web pages by the Microsoft Web Platform. At a fundamental level, any error not trapped in a try/catch block or the `onError` handler applied to a `WinJS.Promise` object (in a call to the `then` or `done` methods, for example) raises the `WinJS.Application.onerror` event. I'll examine that shortly.

In practice, you can listen for errors at other levels in addition to `Application.onerror`. With WinJS and the templates provided by Visual Studio, you can also handle errors at the page-control level and at the navigation level. When an error is raised while the app is navigating to and loading a page, the error triggers the navigation-level error handling, then the page-level error handling, and finally the application-level error handling. You can cancel the error at the navigation level, but any event handlers applied to the page error handler will still be raised.

In this article, I'll take a look at each layer of error handling, starting with the most important: the `Application.onerror` event.

Application-Level Error Handling

WinJS provides the `WinJS.Application.onerror` event (bit.ly/1c0tjC), your app's most basic line of defense against errors. It picks up all errors caught by `window.onerror`. It also catches promises that error out and any errors that occur in the process of managing app model events. Although you can apply an event handler to the `window.onerror` event in your app, you're better off just using `Application.onerror` for a single queue of events to monitor.

As much as users prefer apps that are completely error-free, they are exceptionally good at finding new and creative ways to break apps—ways you never dreamed of.

Once the `Application.onerror` handler catches an error, you need to decide how to address it. There are several options:

- For critical blocking errors, alert the user with a message dialog. A critical error is one that affects continued operation of the app and might require user input to proceed.
- For informational and non-blocking errors (such as a failure to sync or obtain online data), alert the user with a flyout or an inline message.
- For errors that don't affect the UX, silently swallow the error.
- In most cases, write the error to a `tracelog` (especially one that's hooked up to an analytics engine) so you can acquire customer telemetry. For available analytics SDKs, visit the Windows services directory at services.windowsstore.com and click on Analytics (under "By service type") in the list on the left.

Figure 2 Adding a Message Dialog

```
app.onerror = function (err) {  
  
    var message = err.detail.errorMessage ||  
        (err.detail.exception && err.detail.exception.message) ||  
        "Indeterminate error";  
  
    if (Windows.UI.Popups.MessageDialog) {  
        var messageDialog =  
            new Windows.UI.Popups.MessageDialog(  
                message,  
                "Something bad happened ...");  
  
        messageDialog.showAsync();  
        return true;  
    }  
}
```

For this example, I'll stick with message dialogs. I open up default.js (/js/default.js) and add the code shown in **Figure 2** inside the main anonymous function, below the handler for the app.oncheckpoint event.

In this example, the error event handler shows a message telling the user an error has occurred and what the error is. The event handler returns true to keep the message dialog open until the user dismisses it. (Returning true also informs the WWAHost.exe process that the error has been handled and it can continue.)

Now I'll create some errors for this code to handle. I'll create a custom error, throw the error and then catch it in the event handler. For this first example, I add a new folder named handlingErrors to the pages folder. In the folder, I add a new Page Control by right-clicking the project in Solution Explorer and selecting Add | New Item. When I add the handlingErrors Page Control to my project, Visual Studio provides three files in the handlingErrors folder (/pages/handlingErrors): handlingErrors.html, handlingErrors.js and handlingErrors.css.

I open up handlingErrors.html and add this simple markup inside the <section> tag of the body:

```
<!-- When clicked, this button raises a custom error. -->  
<button id="throwError">Throw an error!</button>
```

Next, I open handlingErrors.js and add an event handler to the button in the ready method of the PageControl object, as shown in **Figure 3**. I've provided the entire PageControl definition in handlingErrors.js for context.

Custom Errors

The **Application.onerror** event has some expectations about the errors it handles. The best way to create a custom error is to use the WinJS.ErrorFromName object (bit.ly/1gDESJC). The object created exposes a standard interface for error handlers to parse.

To create your own custom error without using the ErrorFromName object, you need to implement a toString method that returns the message of the error.

Otherwise, when your custom error is raised, both the Visual Studio debugger and the message dialog show "[Object object]." They each call the toString method for the object, but because no such method is defined in the immediate object, it goes through the chain of prototype inheritance for a definition of toString. When it reaches the Object primitive type that does have a toString method, it calls that method (which just displays information about the object).

Figure 3 Definition of the handlingErrors PageControl

```
// For an introduction to the Page Control template, see the following  
documentation:  
// http://go.microsoft.com/fwlink/?LinkId=232511  
(function () {  
    "use strict";  
  
    WinJS.UI.Pages.define("/pages/handlingErrors/handlingErrors.html", {  
        ready: function (element, options) {  
  
            // ERROR: This code raises a custom error.  
            throwError.addEventListener("click", function () {  
  
                var newError = new WinJS.ErrorFromName("Custom error", "I'm an error!");  
                throw newError;  
  
            });  
  
        },  
  
        unload: function () {  
            // Respond to navigations away from this page.  
        },  
  
        updateLayout: function (element) {  
            // Respond to changes in layout.  
        }  
    });  
})();
```

Now I press F5 to run the sample, navigate to the handlingErrors page and click the "Throw an error!" button. (If you're following along, you'll see a dialog box from Visual Studio informing you an error has been raised. Click Continue to keep the sample running.) A message dialog then pops up with the error, as shown in **Figure 4**.

Page-Level Error Handling

The PageControl object in WinJS provides another layer of error handling for an app. WinJS will call the IPageControlMembers.error method when an error occurs while loading the page. After the page has loaded, however, the IPageControlMembers.error method errors are picked up by the Application.onerror event handler, ignoring the page's error method.

WinJS provides the
WinJS.Application.onerror event,
your app's most basic line of
defense against errors.

I'll add an error method to the PageControl that represents the handleErrors page. The error method writes to the JavaScript console in Visual Studio using WinJS.log. The logging functionality needs to be started up first, so I need to call WinJS.Utilities.startLog before I attempt to use that method. Also note that I check for the existence of the WinJS.log member before I actually call it.

The complete code for handleErrors.js (/pages/handleErrors/handleErrors.js) is shown in **Figure 5**.

Now I'll try running the sample and clicking "Throw an error!" again. This results in the exact same behavior as before: Visual Studio picks up the error and then the Application.onerror event fires.

DEVELOPED FOR INTUITIVE USE

DynamicPDF—Comprehensive PDF Solutions for .NET Developers

ceTe Software's DynamicPDF products provide real-time PDF generation, manipulation, conversion, printing, viewing, and much more. Providing the best of both worlds, the object models are extremely flexible but still supply the rich features you need as a developer. Reliable and efficient, the high-performance software is easy to learn and use. If you do encounter a question with any of our components, simply contact ceTe Software's readily available, industry-leading support team.



DynamicPDF

[WWW.DYNAMICPDF.COM](http://www.DynamicPDF.com)



TRY OUR PDF SOLUTIONS FREE TODAY!

www.DynamicPDF.com/eval or call 800.631.5006 | +1 410.772.8620

ceTe software

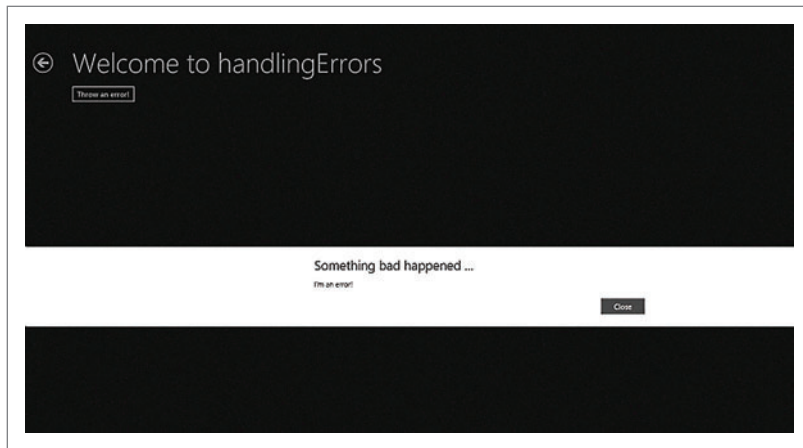


Figure 4 The Custom Error Displayed in a Message Dialog

The JavaScript console doesn't show any messages related to the error because the error was raised after the page loaded. Thus, the error was picked up only by the `Application.onerror` event handler.

So why use the `PageControl` error handling? Well, it's particularly helpful for catching and diagnosing errors in WinJS controls that are created declaratively in the HTML. For example, I'll add the following HTML markup inside the `<section>` tags of `handleErrors.html` (`/pages/handleErrors/handleErrors.html`), below the button:

```
<!-- ERROR: AppBarCommands must be button elements by default
      unless specified otherwise by the 'type' property. -->
<div data-win-control="WinJS.UI.AppBarCommand"></div>
```

Now I press F5 to run the sample and navigate to the `handleErrors` page. Again, the message dialog appears until dismissed. However, the following message appears in the JavaScript console (you'll need to switch back to the desktop to check this):

```
pageError: Page: Invalid argument: For a button, toggle, or flyout
command, the element must be null or a button element
```

Note that the app-level error handling appeared even though I handled the error in the `PageControl` (which logged the error). So how can I trap an error on a page without having it bubble up to the application?

WinJS.log

The call to `WinJS.Utilities.startLog` shown in **Figure 5** starts the `WinJS.log` helper function, which writes output to the JavaScript console by default. While this helps greatly during design time for debugging, it doesn't allow you to capture error data after users have installed the app.

For apps that are ready to be published and deployed, you should consider creating your own implementation of `WinJS.log` that calls into an analytics engine. This allows you to collect telemetry data about your app's performance so you can fix unforeseen bugs in future versions of your app. Just make sure customers are aware of the data collection and that you clearly list what data gets collected by the analytics engine in your app's privacy statement.

Note that when you overwrite `WinJS.log` in this way, the `WinJS.log` function will catch all output that would otherwise go to the JavaScript console, including things like status updates from the Scheduler. This is why you need to pass a meaningful name and type value into the call to `WinJS.Utilities.startLog` so you can filter out any errors you don't want.

The best way to trap a page-level error is to add error handling to the navigation code. I'll demonstrate that next.

Navigation-Level Error Handling

When I ran the previous test where the `app.onerror` event handler handled the page-level error, the app seemed to stay on the homepage. Yet, for some reason, a Back button control appeared. When I clicked the Back button, it took me to a (disabled) `handlingErrors.html` page.

This is because the navigation code in `navigator.js` (`/js/navigator.js`), which is provided in the Navigation App project template, still attempts to navigate to the page even though the page has fizzled. Furthermore, it navigates back to the home-

page and adds the error-prone page to the navigation history. That's why I see the Back button on the homepage after I've attempted to navigate to `handlingErrors.html`.

To cancel the error in `navigator.js`, I replace the `PageControl.Navigator._navigating` function with the code in **Figure 6**. You see that the navigating function contains a call to `WinJS.UI.Pages.render`, which returns a Promise object. The render method attempts to create a new `PageControl` from the URI passed to it and insert it into a host element. Because the resulting `PageControl` contains an error, the returned promise errors out. To trap the error raised during navigation, I add an error handler to the `onError` parameter of the `then` method exposed by that Promise object. This effectively traps the error, preventing it from raising the `Application.onerror` event.

Figure 5 The Complete `handleErrors.js`

```
(function () {
    "use strict";

    WinJS.UI.Pages.define("/pages/handleErrors/handleErrors.html", {

        ready: function (element, options) {

            // ERROR: This code raises a custom error.
            throwError.addEventListener("click", function () {
                var newError = {
                    message: "I'm an error!",
                    toString: function () {
                        return this.message;
                    }
                };
                throw newError;
            });
        },

        error: function (err) {
            WinJS.Utilities.startLog({ type: "pageError", tags: "Page" });
            WinJS.log && WinJS.log(err.message, "Page", "pageError");
        },

        unload: function () {
            // TODO: Respond to navigations away from this page.
        },

        updateLayout: function (element) {
            // TODO: Respond to changes in layout.
        }
    });
})();
```



GdPicture.NET Ultimate | from **\$4,600.56**

All-in-one AnyCPU document-imaging and PDF toolkit for .NET and ActiveX.

- Document viewing, processing, printing, scanning, OMR, OCR, Barcode Recognition
- Annotate image and PDF within your Windows & Web applications
- Read, write and convert vector & raster images in more than 90 formats, including PDF
- Color detection engine for image and PDF compression
- 100% royalty-free and world leading Imaging SDK



ComponentOne Studio Enterprise 2013 v3 | from \$1,315.60

.NET Tools for the Professional Developer: Windows, HTML5/Web, and XAML.

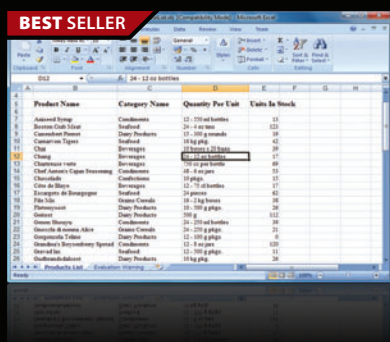
- Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers
- Visual Studio 2013 and Bootstrap support
- New advanced theming tools for WinForms and ASP.NET
- 40+ UI widgets built with HTML5, jQuery, CSS3, and SVG
- New Windows Store Sparkline, DropDown, & Excel controls



Help & Manual Professional | from **\$583.10**

Easily create documentation for Windows, the Web and iPad.

- Powerful features in an easy accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to HTML, WebHelp, CHM, PDF, ePUB, RTF, e-book or print
- Styles and Templates give you full design control



Aspose.Total for .NET | from \$2,449.02

Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, Project plans, emails, barcodes, OCR, and document management in .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from PDF files

Figure 6 The `PageControlNavigator._navigating` Function in `navigator.js`

```
// Other PageControlNavigator code ...

// Responds to navigation by adding new pages to the DOM.
_navigating: function (args) {
    var newElement = this._createPageElement();
    this._element.appendChild(newElement);

    this._lastNavigationPromise.cancel();

    var that = this;
    this._lastNavigationPromise = WinJS.Promise.as().then(function () {
        return WinJS.UI.Pages.render(args.detail.location, newElement,
            args.detail.state);
    }).then(function parentElement(control) {
        var oldElement = that.pageElement;
        // Cleanup and remove previous element
        if (oldElement.winControl) {
            if (oldElement.winControl.unload) {
                oldElement.winControl.unload();
            }
            oldElement.winControl.dispose();
        }
        oldElement.parentNode.removeChild(oldElement);
        oldElement.innerText = "";
    },
    // Display any errors raised by a page control,
    // clear the backstack, and cancel the error.
    function (err) {

        var messageDialog =
            new Windows.UI.Popups.MessageDialog(
                err.message,
                "Sorry, can't navigate to that page.");

        messageDialog.showAsync()

        nav.history.backStack.pop();

        return true;
    });

    args.detail.setPromise(this._lastNavigationPromise);
},

// Other PageControlNavigator code ...
```

Note that it's entirely proper to modify `navigator.js`. Although it's provided by the Visual Studio project template, it's part of your app's code and can be modified however you need.

In the `_navigating` function, I've added an error handler to the final `promise.then` call. The error handler shows a message dialog—as with the application-level error handling—and then cancels the error by returning `true`. It also removes the page from the navigation history.

When I run the sample again and navigate to `handlingErrors.html`, I see the message dialog that informs me the navigation attempt has failed. The message dialog from the application-level error handling doesn't appear.

Promises in WinJS

Creating promises and chaining promises—and the best practices for doing so—have been covered in many other places, so I'll skip that discussion in this article. If you need more information, check out the blog post by [Kraig Brockschmidt](http://bit.ly/1cgManu) at bit.ly/1cgManu or Appendix A in his free e-book, "Programming Windows Store Apps with HTML, CSS, and JavaScript, Second Edition" (bit.ly/1dZwW1k).

Figure 7 The Contents of the `PageControl.ready` Function in `chainedAsync.js`

```
startChain.addEventListener("click", function () {
    goodPromise().
        then(function () {
            return goodPromise();
        }).
        then(function () {
            return badPromise();
        }).
        then(function () {
            return goodPromise();
        }).
        done(function () {
            // This *shouldn't* get called
        },
        function (err) {
            document.getElementById('output').innerText = err.toString();
        });
});
```

Tracking Down Errors in Asynchronous Chains

When building apps in JavaScript, I frequently need to follow one asynchronous task with another, which I address by creating promise chains. Chained promises will continue moving along through the tasks, even if one of the promises in the chain returns an error. A best practice is to always end a chain of promises with a call to the `done` method. The `done` function throws any errors that would've been caught in the error handler for any previous `then` statements. This means I don't need to define error functions for each promise in a chain.

Always end a chain of promises
with a call to the `done` method.

Even so, tracking errors down can be difficult in very long chains once they're trapped in the call to `promise.done`. Chained promises can include multiple tasks, and any one of them could fail. I could set a breakpoint in every task to see where the error pops up, but that would be terribly time-consuming.

Here's where Visual Studio 2013 comes to the rescue. The Tasks window (introduced in Visual Studio 2010) has been upgraded to handle asynchronous JavaScript debugging as well. In the Tasks window you can view all active and completed tasks at any given point in your app code.

Figure 8 The Definitions of the `goodPromise` and `badPromise` Functions in `chainAsync.js`

```
function goodPromise() {
    return new WinJS.Promise(function (comp, err, prog) {
        try {
            comp();
        } catch (ex) {
            err(ex)
        }
    });
}

// ERROR: This returns an errored-out promise.
function badPromise() {

    return WinJS.Promise.wrapError("I broke my promise :(");
}
```


We didn't invent the Internet...

...but our components help you power the apps that bring it to business.



TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**
AS2, EDI/X12, NAESB, OFTP ...
- **Credit Card Processing**
Authorize.Net, TSYS, FDMS ...
- **Shipping & Tracking**
FedEx, UPS, USPS ...
- **Accounting & Banking**
QuickBooks, OFX ...
- **Internet Business**
Amazon, eBay, PayPal ...
- **Internet Protocols**
FTP, SMTP, IMAP, POP, WebDav ...
- **Secure Connectivity**
SSH, SFTP, SSL, Certificates ...
- **Secure Email**
S/MIME, OpenPGP ...
- **Network Management**
SNMP, MIB, LDAP, Monitoring ...
- **Compression & Encryption**
Zip, Gzip, Jar, AES ...



The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to

any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity
powered by 

To learn more please visit our website →

www.nsoftware.com

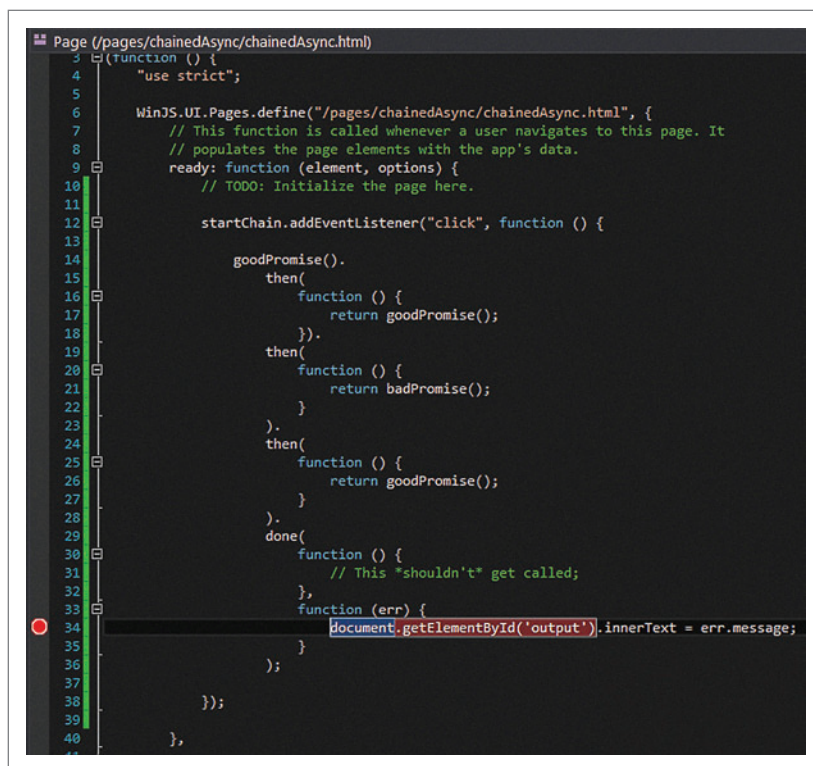


Figure 9 The Position of the Breakpoint in chainedAsync.html

For this next example, I'll add a new page to the solution to demonstrate this awesome tool. In the solution, I create a new folder called chainedAsync in the pages folder and then add a new Page Control named chainAsync.html (which creates /pages/chainedAsync/chainedAsync.html and associated .js and .css files).

In chainedAsync.html, I insert the following markup within the <section> tags:

```
<!-- ERROR:Clicking this button starts a chain reaction with an error. -->
<p><button id="startChain">Start the error chain</button></p>
<p id="output"></p>
```

In chainedAsync.js, I add the event handler shown in Figure 7 for the click event of the startChain button to the ready method for the page.

Last, I define the functions goodPromise and badPromise, shown in Figure 8, within chainAsync.js so they're available inside the PageControl's methods.

I run the sample again, navigate to the "Chained asynchronous" page, and then click "Start the error chain." After a short wait, the message "I broke my promise.:" appears below the button.

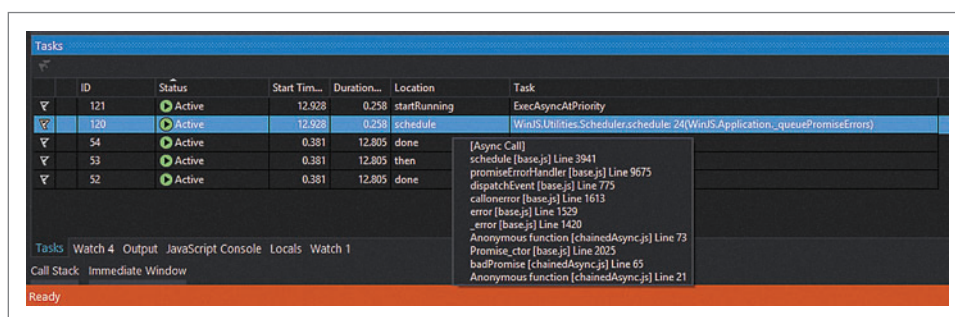


Figure 10 The Tasks Window in Visual Studio 2013 Showing the Error

Now I need to track down where that error occurred and figure out how to fix it. (Obviously, in a contrived situation like this, I know exactly where the error occurred. For learning purposes, I'll forget that badPromise injected the error into my chained promises.)

To figure out where the chained promises go awry, I'm going to place a breakpoint on the error handler defined in the call to done in the click handler for the startChain button, as shown in Figure 9.

I run the same test again, and when I return to Visual Studio, the program execution has stopped on the breakpoint. Next, I open the Tasks window (Debug | Windows | Tasks) to see what tasks are currently active. The results are shown in Figure 10.

At first, nothing in this window really stands out as having caused the error. The window lists five tasks, all of which are marked as active. As I take a closer look, however, I see that one of the active tasks is the Scheduler queuing up promise errors—and that looks promising (please excuse the bad pun).

(If you're wondering about the Scheduler, I encourage you to read the next article in this series, where I'll discuss the new Scheduler API in WinJS.)

When I hover my mouse over that row (ID 120 in Figure 10) in the Tasks window, I get a targeted

view of the call stack for that task. I see several error handlers and, lo and behold, badPromise is near the beginning of that call stack. When I double-click that row, Visual Studio takes me right to the line of code in badPromise that raised the error. In a real-world scenario, I'd now diagnose why badPromise was raising an error.

WinJS provides several levels of error handling in an app, above and beyond the reliable try-catch-finally block. A well-performing app should use an appropriate degree of error handling to provide a smooth experience for users. In this article, I demonstrated how to incorporate app-level, page-level and navigation-level error handling into an app. I also demonstrated how to use some of the new tools in Visual Studio 2013 to track down errors in chained promises.

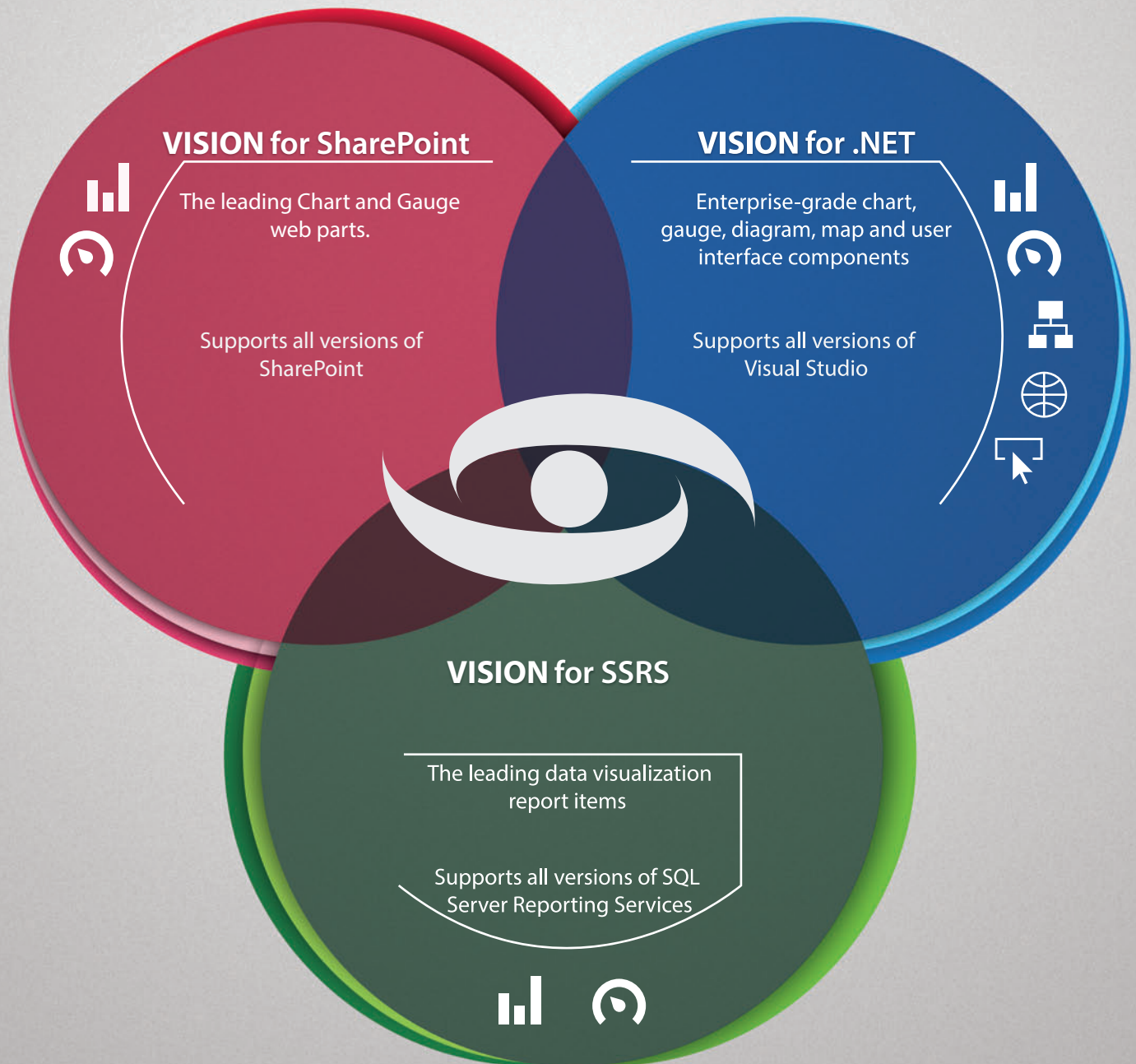
In the next article in this series, I'll explore some of the techniques for making Windows Store apps perform better. I'll examine Web workers, the new Scheduler API in WinJS 2.0 and the new dispose pattern in WinJS 2.0.

ERIC SCHMIDT is a content developer in the Microsoft Windows Developer Content team, writing about the Windows Library for JavaScript (WinJS). He previously worked in the Microsoft Office Division, where he built code samples for the Apps for Office platform. He spends time with his family, plays string bass, builds HTML5 video games and blogs about plastic building toys (historybricks.com).

THANKS to the following Microsoft technical experts for reviewing this article: Kraig Brockschmidt and Josh Williams

Nevron Data Visualization

Visualize Your Success



Nevron components integrate seamlessly in Web and Desktop .NET applications, SQL Server Reporting Services reports and SharePoint portals. All data visualization tools deliver an unmatched set of enterprise - grade features, which makes Nevron the trusted vendor for many Fortune 500 companies.

Download your free evaluation copy from **www.nevron.com** today.

www.nevron.com | email@nevron.com | +1 888-201-6088 (Toll free, USA and Canada)

Microsoft, .NET, ASP.NET, SharePoint, SQL Server and Visual Studio are registered trademarks of Microsoft Corporation in the United States and/or other countries.
Some Nevron components are only available for certain platforms. For details visit **www.nevron.com** or send e-mail to **support@nevron.com**.

Introducing Visual Studio Online

Ed Blankenship

Whether you're part of a team or you're a team of one, with Visual Studio Online you can easily plan, create, construct, build, test, and monitor seriously demanding applications, from anywhere. You don't need a large infrastructure team, and you don't need to touch a single server. As someone who has performed hundreds of Team Foundation Server (TFS) installations and upgrades as a consultant, I love how the drudgery of that kind of routine maintenance is now a thing of the past. Visual Studio Online is updated with the newest features automatically and continuously, so you can focus on what you need to do most: construct your applications!

What's New?

I'm often asked, "Isn't Visual Studio Online just TFS in the cloud?" The answer is yes and no.

TFS was introduced eight years ago, and began the move to a cloud-hosted application lifecycle management (ALM) service in 2012 with the launch of Team Foundation Service. There has been great adoption from day one, with individual development teams starting up and even midsize companies choosing to leave their on-premises infrastructure behind. The big deal is this: every few weeks, new features appear to your team as the TFS product group wraps them up in their internal development iterations. Account holders can find out what's new by watching the Features

Timeline at bit.ly/17DV8YI or by following social media like VSONline on Twitter (twitter.com/vsonline).

So, yes, Visual Studio Online is the next evolution of TFS and Team Foundation Service, bringing you the fundamentals of ALM in the cloud. And no, it's not exactly the same thing.

As Microsoft transforms to a devices and services company, many of you have asked, "What does this mean to me as a developer?" With Visual Studio Online, Microsoft now has a platform of services you can take advantage of whether you're moonlighting on your own personal projects or working on larger systems with your team. I'll explore a few of those services.

ALM and Beyond

Being cloud-based enables Visual Studio Online to deliver some high-powered experiences:

Build in the cloud. No servers, no overhead—great. Your code starts from a clean server image every time the build executes. Those dependencies and machine configurations you didn't even know about causing headaches down the road? Gone. You still have full access to your build drops because they're also stored in your source repository in Visual Studio Online. Need a more complex setup? Consider a hybrid solution—save your precious on-premises resources for those highly customized workflows or infrastructure and use Visual Studio Online for basics such as continuous integration.

Load test in the cloud. It's hard for individuals, startups and even larger teams to do proper performance and load testing because the setup and resources are so expensive and time-consuming. How can you justify the expense when you don't know whether the application you've built will catch on? Instead of launching and hoping, now you have access to full-featured load testing as a service that runs on Microsoft infrastructure, which always has the latest version of the load-testing software. You aren't limited to those retired machines you scrounged from the supply room or stuck figuring out what you need to install and configure—you can set up and scale your tests in whatever way you need to make your run realistic. Whether you're adding 100 virtual users or you need

This article discusses Visual Studio Online Application Insights, which is currently in limited preview.

This article discusses:

- Developing with Visual Studio Online
- Setting up your first iteration
- Charting work items
- On-demand load testing
- Collecting telemetry data

Technologies discussed:

Visual Studio Online, Team Foundation Server, Team Foundation Service, Windows 8, Windows Azure

MAY THE CODE BE WITH YOU



```
History
var history = new History();
var messages = history.NewMessages(10);
messages.Sorting = SortBy.NewestFirst;
messages.Attachments = new List<Attachment>();
messages.Attachments.Add(new Attachment() { Name = "1.jpg", Size = 1000000 });
messages.Attachments.Add(new Attachment() { Name = "2.jpg", Size = 1000000 });
```



Intense Take-Home Training for
Developers, Software Architects
and Designers

Topics include:

- Visual Studio/.NET
- Windows 8.1/WinRT
- WPF/Silverlight
- Cloud Computing
- Windows Phone
- Cross-Platform Mobile Development
- ASP.NET
- HTML5
- Mobile Web
- JavaScript



WANT TO LEARN MORE?

SCAN THIS
QR CODE
TO JOIN OUR
MAILING LIST





Visual Studio Live! Las Vegas is part of Live! 360 DEV, which means you'll have access to four (4) other co-located events at no additional cost:

Five (5) events means over a hundred sessions to choose from – mix and match sessions to create your own, custom event line-up – it's like no other dev conference available today!

live360events.com/lasvegas

SQL Server **LIVE!**
SQL SERVER FOR MODERN DEVELOPERS

SharePoint **LIVE!**
TRAINING FOR COLLABORATION

Modern Apps **LIVE!**
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

AND INTRODUCING Web Dev Live!

Web Dev **LIVE!**
HTML5, JAVASCRIPT & ASP.NET TRAINING

**REGISTER BY
JANUARY 22
AND SAVE \$400!**

vslive.com/lasvegas

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

in linkedin.com – Join the
"Visual Studio Live" group!



Use promo code VSLJANTI

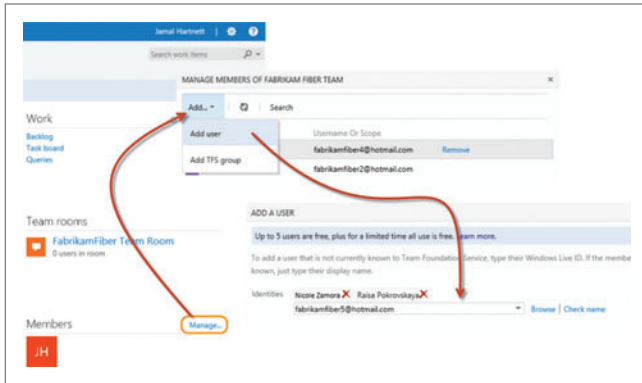


Figure 1 Adding Team Members

to crank it up to get ready for the holiday season, the load-testing service can handle what you need, when you need it.

Real-time, 360-degree application monitoring. You don't have a full-time service center monitoring the status and health of your production systems? With Visual Studio Online, now you do. Visual Studio Online Application Insights collects rich operational, performance and customer usage information from your applications—whether they run on-premises, in Windows Azure, at third-party cloud providers, or in a hybrid manner across all three. More than just keeping the lights on, you can also proactively monitor usage to help you decide which killer feature to build next. The service also takes advantage of application meta-information that can help when you're investigating live issues such as build and deployment information, which lets you get back to your solution and source code in any of your environments (including production). It already knows because it's in the same data store with the rest of your ALM information, which saves you time.

Bring your own IDE, or get a lightweight code editor in your browser. Visual Studio Online makes it easy for you to update your code any way, anytime, anywhere. You have Visual Studio, of course, and now you can use a true Git repository for seamless connection to Eclipse, Xcode, and many other IDEs for your cross-platform development projects. Otherwise, keep it simple: update on the fly, from any modern browser, with the new, lightweight Visual Studio Online code editor, code-named "Monaco." This code editor is complementary with your use of Visual Studio on your desktop.

Just like you, Microsoft is committed to delivering continuous value, with significant new releases to Visual Studio Online every

three weeks. If you have a great idea for a feature, you can even cast your own vote on UserVoice. So what are you waiting for? Head over to visualstudio.com and get started. It's free for the first five users and an included benefit for MSDN subscriptions, and there are introductory discounts for pay-as-you-go services and additional Visual Studio Online user plans for your other team members. You don't have to worry about additional storage, transactions, bandwidth, compute cycles and so on because they're included in the user plans and pay-as-you-go services.

From Zero to Hero: Set up Your Project for Success

Want to see how Visual Studio Online can help you deliver the next killer app? The following scenario will guide you through an example initial iteration with Visual Studio Online. If you've used TFS or Team Foundation Service, some of this will look familiar.

Here's the scenario: You're the lead developer on a small team with limited resources but a great idea for an application. You need to get started quickly, but you expect the app to be around for a while so you want to put your project on a path to continued success. You don't want to waste any time setting up infrastructure or deployment pipelines or test harnesses. You just want to build your application and get it into your customers' hands as soon as possible.

It's quick, easy and painless to get started. Go to visualstudio.com, find the "Get started for free" link in the upper-right, fill out the form and ... you're done.

Once you have an account, you need a place where you and your team can work. In Visual Studio Online, the top-level container is a Team Project, so your first step is to create one of those. You need to make two important decisions at this point:

1. Which kind of source control repository to use, and
2. Which process template would be best for tracking your work items.

The choice for source control is between distributed version control with Git and the traditional version control that you're familiar with in TFS. For example, suppose your team is currently writing a Windows Phone 8 app and its back-end services, but plans eventually to port the mobile app to iPhone and Android devices. Knowing you're going to have to work across multiple platforms and with different IDEs, you choose to go with Git for your source control repository. For more about the choice between Team Foundation Version Control (TFVC) and Git, check out the Channel 9 video at bit.ly/16XPcUK.

The process template is what defines your workflow: what your work items are called, what states they go through as you work on them and other metadata. You can choose between three templates: Scrum, Agile and Capability Maturity Model Integration (CMMI). If your team is doing CMMI or a more formal process, you should probably go with that template. If you're an agile team, or you don't have a particular process, choose between the Agile and Scrum templates. The biggest difference between the two is how they handle bugs and what name they use for backlog items. The Agile template treats bugs as tasks, while the Scrum template displays them on your backlog along with other backlog items. If you aren't sure, stick with the Scrum template—it's the default for a reason. Choosing the Scrum template doesn't mean you have to do Scrum; it simply means you'll see Scrum terminology used for

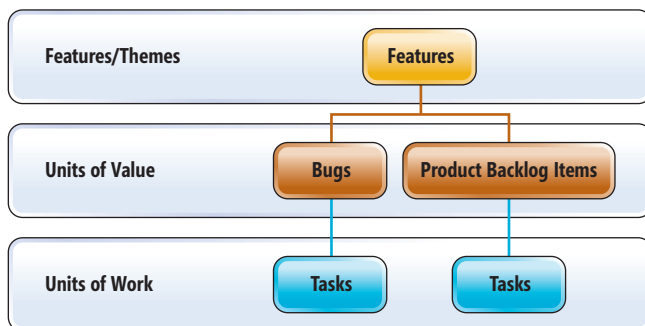


Figure 2 Work Item Relationships

work item types. Scrum refers to requirements as Product Backlog Items (PBIs) and the Agile template refers to them as User Stories. Learn more about working with team projects and the key differences between the process templates at bit.ly/Vh0azl.

Once you have your account and your team project, it's time to add your team members. To add them, make sure you have their Microsoft accounts (formerly Live IDs), browse to your project homepage (myawesometeam.visualstudio.com) and follow the flow shown in **Figure 1**.

If you want to fine-tune permissions and other administrative details, use the settings gear icon at the top.

Get on Track: Set up Your First Iteration

With the administrative details complete, it's time to get to work. You've chosen the Scrum template, but your team (all five of you) is not completely sold on the need to adopt formal Scrum. You know you want to work in iterations, delivering functioning code as fast as possible and incorporating feedback from your customers into the next iteration. You also want to keep track of what work is currently in progress and what work is coming up, and have a way to prioritize that work.

Before getting started with work items, I'll take a quick look at some of the details. Choosing the Scrum template means you'll be working primarily with features, PBIs, bugs and tasks (there are other work item types, but most of them help you behind the scenes). **Figure 2** outlines the relationships between these work item types, which are maintained with work item links.

Generally, a feature represents an amount of work that will span iterations (these are sometimes referred to as epics), PBIs are units of customer value that can be completed in a single iteration, and tasks are units of work that can be completed in a couple of days. Using them this way lets you scope your project view to whatever level of granularity makes sense at the time, whether at the portfolio level or down to the details. There's a lot more to work items than what I discuss here, and I encourage you to check out the relevant articles on MSDN for more information.

One final note on working with work items: just because a field is there doesn't mean you have to use it. Use the fields that make sense for your team and ignore those that do not. My advice from helping many teams in the past is to keep things as simple as possible.

It's time to start filling the backlog, so click the Work tab on the project page. This will bring up your product backlog.

The menu on the left lets you scope your view to the appropriate level of granularity, and you can add items to that level from the list on the right. Reorder your PBIs vertically to prioritize your backlog.

Now that you have some work in the Product Backlog, it's time to start thinking about your first iteration. Your team wants to move fast, so it decides to deliver iterations in one week, with the goal of going to production every Tuesday. The first thing to do is give your iteration some dates. You can accomplish this in the settings menu or, if your view is scoped to a particular iteration, in the "Set Dates" link.

After your iteration has dates assigned to it, you can start assigning PBIs from your backlog into your iteration by dragging them from the right and dropping them on the appropriate iteration on the left. Once the PBIs are in an iteration you can break them into individual tasks for your team members, as shown in **Figure 3**.

This is just the tip of the iceberg regarding what you can do with the agile planning tools. As your team grows and your needs become more complex, you can add things like

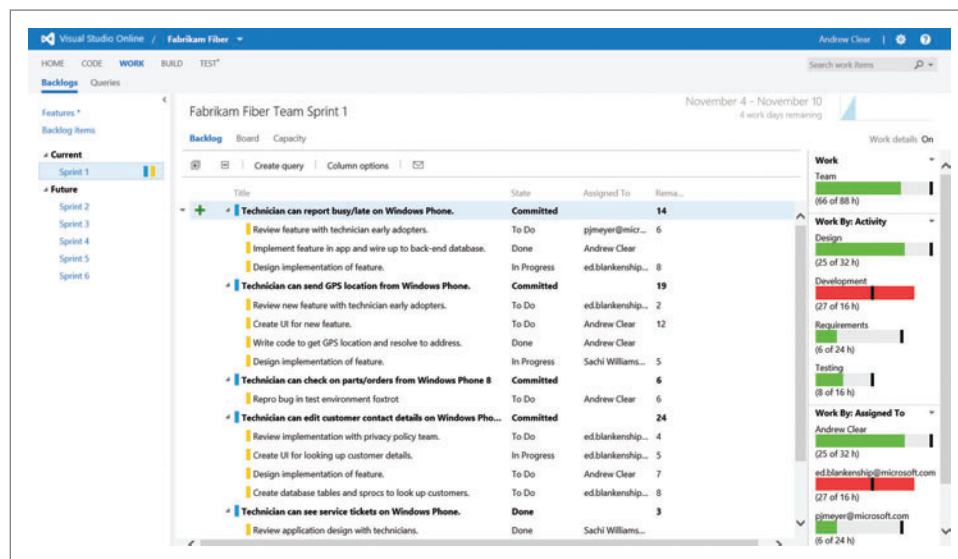


Figure 3 Iteration Planning

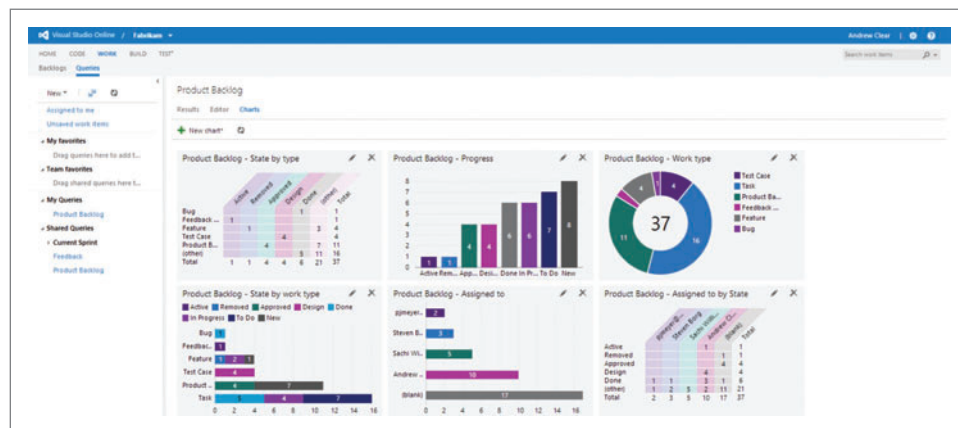


Figure 4 Work Item Charts

WORKFLOW APPLICATIONS | HELP DESK | BUG TRACKING MADE EASY!

Alexsys Team[®] offers *flexible task management* software for Windows, Web, and Mobile Devices.
Track whatever you need to get the job done - anywhere, anytime on practically any device!



Thousands are using Alexsys Team[®] to create workflow solutions and web apps - without coding!
Fortune 500 companies, state, local, DOD, and other federal agencies use Team to manage their tasks.
Easily tailor Team to meet your exact requirements - even if they change daily, weekly, or even every minute.

Alexsys Team[®] Features Include:

- Form and Database customization
- Custom workflows
- Role-based security
- Adaptive Rules Engine
- DOD CAC card support
- Time recording
- Automated Escalations, Notifications, and SOAP Messaging
- Supports MS-SQL, MySQL, and Oracle databases

Our renowned tech support team is here to make you and your team a success. Don't have enough resources? Our professional services staff has helped companies large and small use Alexsys Team[®] at a fraction of the cost of those big consulting firms.

Find out yourself:

Free Trial and Single User FreePack™
available at Alexcorp.com



FIND OUT MORE!

1-888-880-ALEX (2539)
ALEXCORP.COM

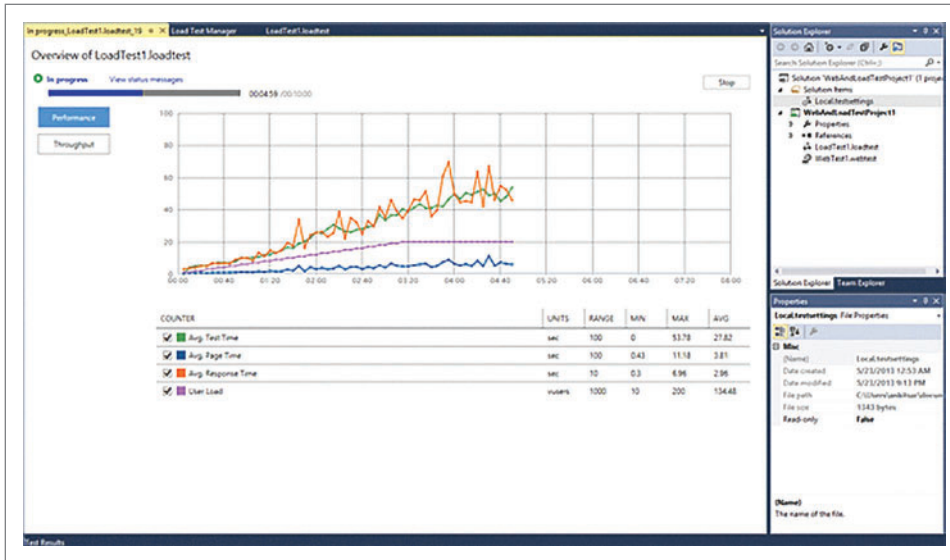


Figure 5 Viewing Load-Test Results

capacity planning, forecasting and test-plan management. For now, this is as much complexity as you need, so start completing PBIs.

As you burn through items in your backlog, you need to keep track of your work in progress. Your team wants to get a better feel for the status of the current iteration, so during your next daily standup meeting you pull up the iteration task board.

This gives you a good view into the status of each PBI and helps keep your daily standup meeting on track. The iteration task board is drag-and-drop and touch-enabled, so you can quickly make updates during the meeting to keep the board (and your work item states and team assignments) up to date.

The iteration task board is great for tracking work, but to get a feel for the flow of value through your team you need a board scoped to the PBI or feature level. Switch over to the Kanban board, which is accessible from the product backlog.

From here, you can set work-in-progress limits for each column, track the flow of work, and check out your team's cumulative flow

and easily produce the reports you need.

Ten minutes later, your investor is happy and you can get back to making your customers happy. You can even pin the work item charts you've created to the team's dashboard.

Delivering Value: Building Your Application

You know how to code. You also know that keeping quality up to snuff is of paramount importance to your customers. Unfortunately, you and your team are so full of new ideas that sometimes it's easy to race on to the next notion, without fully fleshing out the current one.

Your code base doesn't always compile successfully. After a quick retrospective, your team decides it needs to start using automated builds, either by using a continuous integration (CI) build or even putting a gated check-in in place to keep bad code out of the repository. You don't have the infrastructure or the time to set up your own on-premises build server. That's where the Visual Studio Online cloud build service comes to the rescue.

From Team Explorer in your Visual Studio IDE, select the build page and then New Build Definition. Give the new definition a name, select your trigger and walk through the rest of the wizard. Don't forget to save your build definition when you're done.

Just because it's a CI build (or any other trigger type) doesn't mean you can't kick it off manually whenever you need a build off-schedule. Of course, you could kick it off from Team Explorer, but where's the fun in that? Pull up your friendly browser and head to the

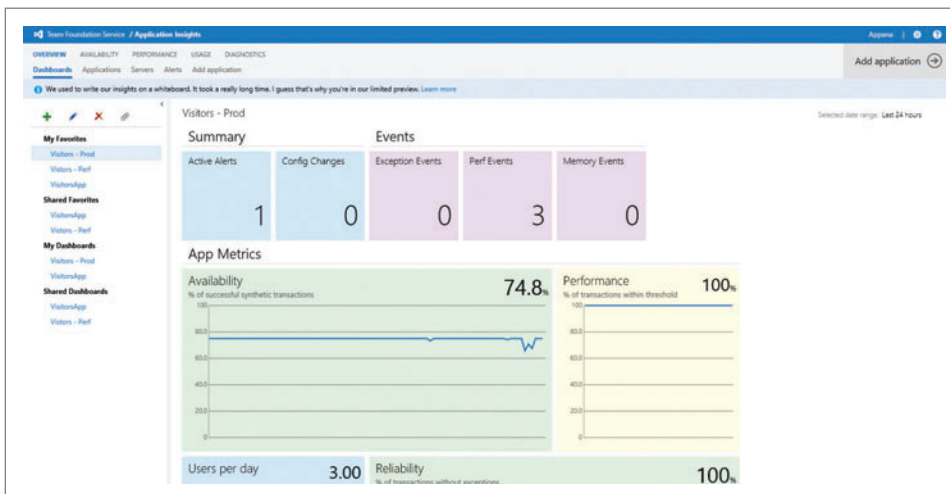


Figure 6 Application Insights Dashboard

Creating a report is as easy as writing a letter



Reuse MS Word documents as your reporting templates



Create encrypted and print-ready Adobe PDF and PDF/A



Royalty-free WYSIWYG template designer



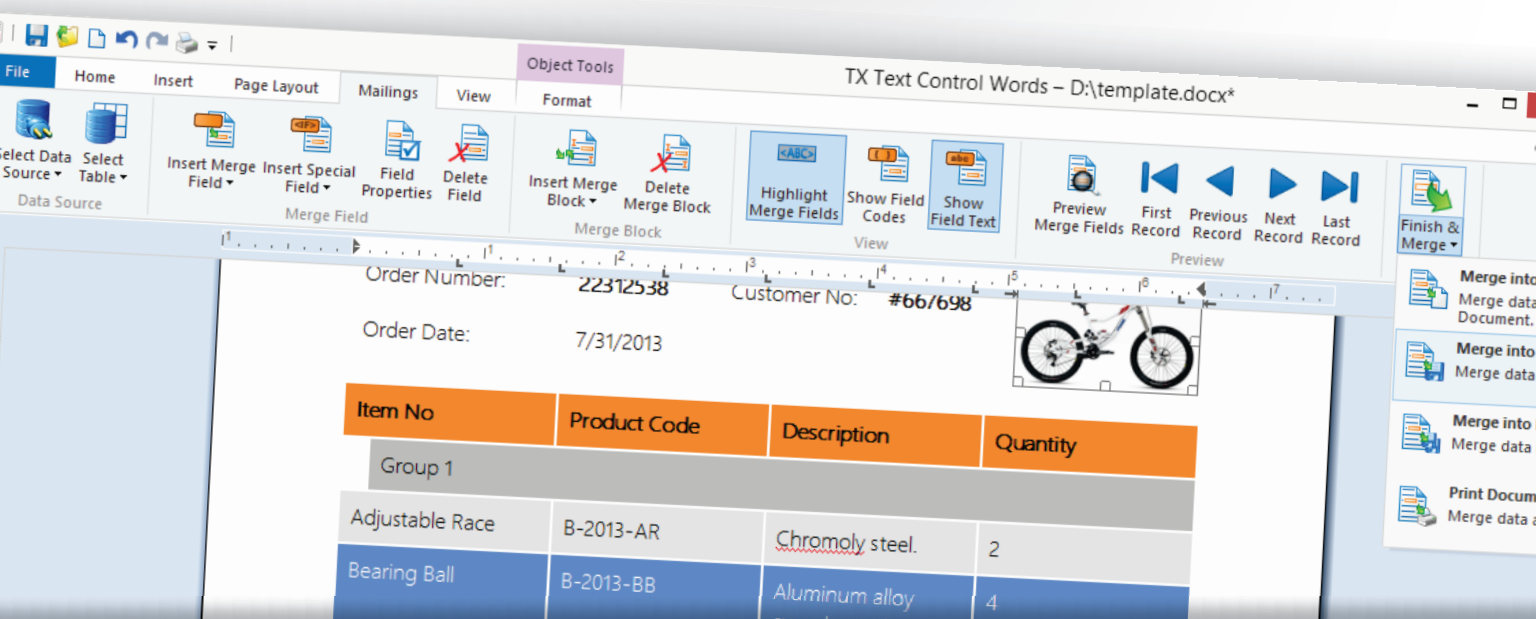
Powerful and dynamic 2D/3D charting support



Easy database connections and master-detail nested blocks



1D/2D barcode support including QRCode, IntelligentMail, EAN



www.textcontrol.com/reporting



txtextcontrol

US: +1 855-533-TEXT
EU: +49 421 427 067-10



Visual Studio

Microsoft

Partner

Reporting

Rich Text Editing

Spell Checking

Barcodes

PDF Reflow

Build tab on your project page. Select your build from the list on the left and click Queue Build.

After you have a CI build set up and underway, jump over to your team room to view the results and share the good news with your team. For more information on builds, check out bit.ly/158otoM.

Load Testing on Demand

Your user base is growing rapidly, but users are reporting a lot of lag on your site. You're unable to pin down the source of the lag, so you want to create some realistic performance and load tests to help uncover the slow spots. You begin by recording some Web performance tests with Visual Studio Ultimate. Each test represents a typical way a user would interact with your site, which is called a synthetic transaction.

You then combine the performance tests to simulate a typical load on the system. Later on, you plan to create some other scenarios, but for now, you just want to stick to the normal happy-path scenario.

Once you've balanced your performance tests into a load test, you set the load test to run on Visual Studio Online by opening the test settings file and choosing Visual Studio Online. You can then watch as the data rolls in, as shown in **Figure 5**.

Yes, you did just do that in less than an hour. For a more detailed walk-through, see bit.ly/18hlcCm.

Monitor and Continuously Learn: Collect Telemetry Data

Your application and its supporting Windows Azure sites and services are live. Users seem to be downloading it. People seem to be browsing the site and the client app is hitting the services back end. Something, at least, appears to be happening. You're just not quite sure what ...

You need to start collecting telemetry data across your entire application. Not just the Web pages, server performance and Windows Phone 8 app, and not just on your Windows Azure data tier. You need a complete view across the entire end-to-end UX flow. Meet the new Application Insights service on Visual Studio Online.

To get started with the Application Insights service, go to the home-page of your Visual Studio Online account and click on the Application Insights tile. You need to register your application with your invitation code because the service is currently in limited preview. Once that's done, you can set up a simple test to ping a URL (the New Single URL Test) and watch the data start flowing. This is an example of a basic ping test, but you can also leverage the synthetic transactions you created as Web performance tests for your load tests earlier. You can then monitor your application with basic ping tests or synthetic transactions from multiple datacenters from around the world.

You can also configure the Microsoft Monitoring Agent to report application performance-monitoring data to the Application Insights service. This will provide an even better look at how your application's services are behaving with each of its dependencies, including databases and other services.

For most sites and connected apps, nothing is more important than uptime. Your application is no different and, unfortunately, you've been blindsided by some customer reports that your service was down even though you thought it was up. You need live, constant availability monitoring so you're the first to know if a test fails or your site goes down. That's exactly what Application Insights can give you.

Knowing whether your site is available is great. Knowing exactly what line of code or stored procedure is slowing it down is better. Being able to jump directly to that line of code in your IDE from your performance-monitoring service is, well, downright amazing. Application Insights discovers dependencies, shows you when you've made new deployments and aggregates all of the performance information to give you the top 10 slowest parts of your application.

Now that you have performance monitoring from Application Insights, you've drastically driven down your mean time to repair service and your mean time to restore service.

When you first went to market with your application, you had a solid understanding of what your application was going to do and how your customers were going to interact with it. Now your customers' needs are evolving and your application needs to evolve with them. You have many hypotheses about how to improve the customer experience, but what you need is the telemetry data and meta-telemetry data to prove your assumptions and hypotheses. With the inclusion of usage monitoring in Application Insights, you get the multidimensional data that will guide your future investments in your application, as shown in **Figure 6**.

As your team works more and more with Application Insights, you begin to realize you've gone through a paradigm shift in the way you build software. No longer are those insights an afterthought; you've made this process a full member of your application lifecycle, drastically influencing how your team thinks about the application. You can even use the SDK to start reporting custom business metrics that can be tracked on your dashboards, or to send custom developer trace events.

Application Insights has created a connection between your developers and the way customers use your application. Questions of how the user is going to interact with the application, how a new change is going to impact performance and how to maintain the near-perfect availability of the application are discussed throughout the entire lifecycle.

Wrapping Up

So there you have it: a quick introduction to the newest member of the Visual Studio family. Visual Studio Online is a cloud-based platform of tools and services that helps you plan, deploy and manage your application. It evolved from TFS and Team Foundation Service, bringing the Microsoft ALM platform to the cloud and strengthening it with new services like Application Insights and an online IDE. And it's just the beginning. Even if you're using TFS on-premises, you can take advantage of Visual Studio Online services like load testing and Application Insights as you need them. The Visual Studio Online team is committed to continuously delivering value, with significant updates every few weeks. ■

Ed Blankenship is product manager for Visual Studio Online and Visual Studio ALM at Microsoft. Before joining Microsoft, he was voted Microsoft MVP of the Year (Visual Studio ALM, Team Foundation Server) for 2010 and was a Microsoft MVP for five years. He's the lead author of two books on Team Foundation Server, including "Professional Team Foundation Server 2012" (Wrox, 2013).

THANKS to the following technical experts for reviewing this article: Andrew Clear and Cheryl Hammond (ALM consultants with Northwest Cadence)



Extreme Performance Linear Scalability



For .NET & Java Apps



Remove data storage performance bottlenecks and scale your applications to extreme transaction processing (XTP). Cache data in memory and reduce expensive database trips. NCache scales linearly by letting you add inexpensive cache servers at runtime. JvCache is 100% Java implementation of NCache.

Enterprise Distributed Cache

- Extremely fast & linearly scalable with 100% uptime
- Mirrored, Replicated, Partitioned, and Client Cache
- NHibernate & Entity Framework Level-2 Cache

ASP.NET Optimization in Web Farms

- ASP.NET Session State storage
- ASP.NET View State cache
- ASP.NET Output Cache provider
- ASP.NET JavaScript merge/minify

Runtime Data Sharing

- Powerful event notifications for pub/sub data sharing



www.alachisoft.com

Download a **FREE** trial!

1-925-236-3830

Build Fault-Tolerant Composite Applications

Ivan Krivyakov

There's a widespread need for composite applications, but fault-tolerance requirements vary. In some scenarios, it might be OK for a single failing plug-in to bring down the whole application. In other scenarios, this isn't acceptable. In this article, I describe an architecture for a fault-tolerant composite desktop application. This proposed architecture will provide a high level of isolation by running each plug-in in its own Windows process. I built it with the following design goals in mind:

- Strong isolation between the host and plug-ins
- Complete visual integration of plug-in controls into the host window
- Easy development of new plug-ins

This article discusses:

- Composite application alternatives
- Isolating components
- Architecting and developing an example application
- Working with host and plug-in services
- Logging, error handling and versioning

Technologies discussed:

Windows Presentation Foundation, Microsoft .NET Framework

Code download available at:

msdn.microsoft.com/magazine/msdnmag0114

- Reasonably easy conversion of existing applications to plug-ins
- Ability for plug-ins to use services provided by the host, and vice versa
- Reasonably easy addition of new services and interfaces

The accompanying source code (msdn.microsoft.com/magazine/msdnmag0114) contains two Visual Studio 2012 solutions: WpfHost.sln and Plugins.sln. Compile the host first, and then compile the plug-ins. The main executable file is WpfHost.exe. The plug-in assemblies are loaded on demand. **Figure 1** shows the completed application.

Architectural Overview

The host displays a tab control and a "+" button in the top-left corner that shows a list of available plug-ins. The list of plug-ins is read from the XML file named plugins.xml, but alternative catalog implementations are possible. Each plug-in is executed in its own process, and no plug-in assemblies are loaded into the host. A high-level view of the architecture is shown in **Figure 2**.

Internally, the plug-in host is a regular Windows Presentation Foundation (WPF) application that follows the Model-View-View-Model (MVVM) paradigm. The model part is represented by the PluginController class, which holds a collection of loaded plug-ins. Each loaded plug-in is represented by an instance of the Plugin class, which holds one plug-in control and talks to one plug-in process.

The hosting system consists of four assemblies, organized as shown in **Figure 3**.

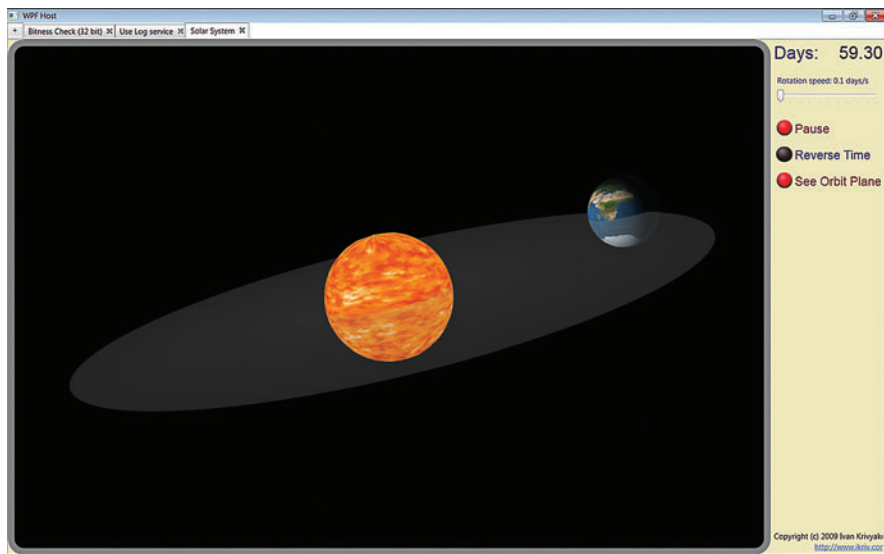


Figure 1 The Host Window Seamlessly Integrates with the Out-of-Process Plug-Ins

WpfHost.exe is the host application. PluginProcess.exe is the plug-in process. One instance of this process loads one plug-in. WpfHost.Interfaces.dll contains common interfaces used by the host, the plug-in process and the plug-ins. PluginHosting.dll contains types used by the host and the plug-in process for plug-in hosting.

Loading a plug-in involves some calls that must be executed on the UI thread, and some calls that can be executed on any thread. To make the application responsive, I only block the UI thread when strictly necessary. Hence, the programming interface for the Plugin class is broken into two methods, Load and CreateView:

```
class Plugin
{
    public FrameworkElement View { get; private set; }
    public void Load(PluginInfo info); // Can be executed on any thread
    public void CreateView();           // Must execute on UI thread
}
```

The Plugin.Load method starts a plug-in process and creates the infrastructure on the plug-in process side. It's executed on a worker thread. The Plugin.CreateView method connects the local view to the remote FrameworkElement. You'll need to execute this on the UI thread to avoid exceptions such as an InvalidOperationException.

The Plugin class ultimately calls a user-defined plug-in class inside the plug-in process. The only requirement for that user class is that it implements the IPlugin interface from the WpfHost.Interfaces assembly:

```
public interface IPlugin : IServiceProvider, IDisposable
{
    FrameworkElement CreateControl();
}
```

The framework element returned from the plug-in may be of arbitrary complexity. It might be a single text box or an elaborate user control that implements some line-of-business (LOB) application.

The Need for Composite Applications

Over the last few years, a number of my clients have expressed the same business need: desktop applications that can load external plug-ins, thus combining several LOB applications under one "roof." The underlying reason for this requirement can vary. Multiple teams might develop different parts of the application

on different schedules. Different business users might require different sets of features. Or maybe the clients want to ensure the stability of the "core" application, while at the same time maintaining flexibility. One way or another, the requirement to host third-party plug-ins has come up more than once within different organizations.

There are several traditional solutions for this problem: the classic Composite Application Block (CAB), the Managed Add-In Framework (MAF), the Managed Extensibility Framework (MEF) and Prism. Another solution was published in the August 2013 issue of MSDN by my former colleagues Gennady Slobodsky and Levi Haskell (see the article, "Architecture for Hosting Third-Party .NET Plug-Ins," at msdn.microsoft.com/magazine/dn342875). These

solutions are all of great value, and many useful applications have been created using them. I'm an active user of these frameworks as well, but there's one problem that kept haunting me for quite some time: stability.

Applications crash. That's a fact of life. Null references, unhandled exceptions, locked files and corrupted databases aren't going to disappear anytime soon. A good host application must be able to survive a plug-in crash and move on. A faulty plug-in must not be allowed to take down the host or other plug-ins. This protection need not be bulletproof; I'm not trying to prevent malicious hacking attempts. However, simple mistakes such as an unhandled exception in a worker thread shouldn't bring the host down.

A good host application must be
able to survive a plug-in crash
and move on.

Isolation Levels

Microsoft .NET Framework applications can handle third-party plug-ins in at least three different ways:

- **No isolation:** Run host and all plug-ins in a single process with a single AppDomain.
- **Medium isolation:** Load each plug-in in its own AppDomain.
- **Strong isolation:** Load each plug-in in its own process.

No isolation entails the least protection and least control. All data is globally accessible, there's no fault protection and there's no way to unload the offending code. The most typical cause of an application crash is an unhandled exception in a worker thread created by a plug-in.

You can try to protect host threads with try/catch blocks, but when it comes to plug-in-created threads, all bets are off. Starting with the .NET Framework 2.0, an unhandled exception in any

thread terminates the process, and you can't prevent this. There's a good reason for such seeming cruelty: An unhandled exception means the application probably has become unstable, and letting it continue is dangerous.

Medium isolation provides more control over a plug-in's security and configuration. You can also unload plug-ins, at least when things are going well and no threads are busy executing unmanaged code. However, the host process still isn't protected from plug-in crashes, as demonstrated in my article, "AppDomains Won't Protect Host from a Failing Plug-In" (bit.ly/1f07sp0). Designing a reliable error-handling strategy is difficult, if not impossible, and the unloading of the failing AppDomain isn't guaranteed.

AppDomains were invented for hosting ASP.NET applications as lightweight alternatives to processes. See Chris Brumme's 2003 blog post, "AppDomains ("application domains")," at bit.ly/PolX1r. ASP.NET applies a relatively hands-off approach to fault tolerance. A crashing Web application can easily bring down the whole worker process with multiple applications. In this case, ASP.NET simply restarts the worker process and reissues any pending Web requests. This is a reasonable design decision for a server process with no user-facing windows of its own, but it might not work as well for a desktop application.

Strong isolation provides the ultimate level of protection against failures. Because each plug-in runs in its own process, the plug-ins can't crash the host, and they can be terminated at will. At the same time, this solution requires a rather complex design. The application has to deal with a lot of inter-process communication and synchronization. It also must marshal WPF controls across process boundaries, which isn't trivial.

As with other things in software development, choosing an isolation level is a trade-off. Stronger isolation gives you more control and more flexibility, but you pay for it with increased application complexity and slower performance.

Some frameworks choose to ignore fault tolerance and work at the "no isolation" level. MEF and Prism are good examples of that approach. In cases where fault tolerance and fine-tuning plug-in configuration aren't issues, this is the simplest solution that works and is therefore the correct one to use.

Many plug-in architectures, including the one proposed by Slobodsky and Haskell, use medium isolation. They achieve isolation via AppDomains. AppDomains give host developers a significant degree of control over plug-in configuration and security. I personally built a number of AppDomain-based solutions over the past several years. If the application requires unloading code, sandboxing and configuration control—and if fault tolerance isn't an issue—then AppDomains are definitely the way to go.

MAF stands out among add-in frameworks because it lets host developers choose any of the three isolation levels. It can run an add-in in its own process using the `AddInProcess` class. Unfortunately, `AddInProcess` doesn't work for visual components out of the box. It might be possible to extend MAF to marshal visual components across processes, but this would mean adding another layer to an already complex framework. Creating MAF add-ins isn't easy, and with another layer on top of MAF, the complexity is likely to become unmanageable.

My proposed architecture aims to fill the void and provide a robust hosting solution that loads plug-ins in their own processes and provides visual integration between the plug-ins and the host.

Strong Isolation of Visual Components

When a plug-in load is requested, the host process spawns a new child process. The child process then loads a user plug-in class that creates a `FrameworkElement` displayed in the host (see **Figure 4**).

The `FrameworkElement` can't be marshaled directly between processes. It doesn't inherit from `MarshalByRefObject`, nor is it marked as `[Serializable]`, so .NET remoting won't marshal it. It isn't marked with the `[ServiceContract]` attribute, so Windows Communication Foundation (WCF) won't marshal it, either. To overcome this problem, I use the `System.Addin.FrameworkElementAdapters` class from the `System.Windows.Presentation` assembly that's part of MAF. This class defines two methods:

- The `ViewToContractAdapter` method converts a `FrameworkElement` to an `INativeHandleContract` interface, which can be marshaled with .NET remoting. This method is called inside the plug-in process.
- The `ContractToViewAdapter` method converts an `INativeHandleContract` instance back to `FrameworkElement`. This method is called inside the host process.

Unfortunately, the simple combination of these two methods doesn't work well out of the box. Apparently, MAF was designed to marshal WPF components between AppDomains and not between processes. The `ContractToViewAdapter` method fails on the client side with the following error:

```
System.Runtime.Remoting.RemotingException:
Permission denied: cannot call non-public or static
methods remotely
```

The root cause is that the `ContractToViewAdapter` method calls the constructor of the class, `MS.Internal.Controls.AddInHost`, which attempts to cast the `INativeHandleContract` remoting proxy to type `AddInHwndSourceWrapper`. If the cast succeeds, then it calls the internal method `RegisterKeyboardInputSite` on the remoting proxy. Calling internal methods on

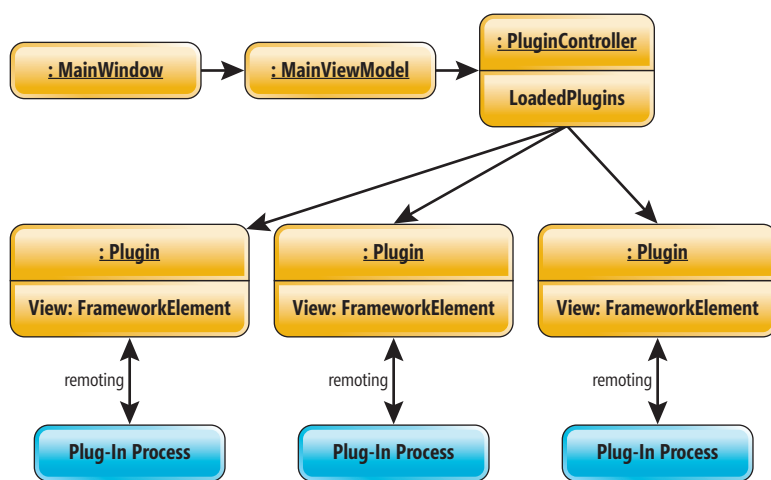


Figure 2 A High-Level View of the Application Architecture

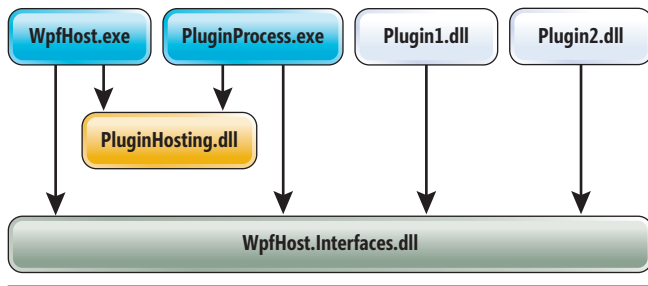


Figure 3 The Assemblies of the Hosting System

cross-process proxies isn't allowed. Here's what's happening inside the AddInHost class constructor:

```

// From Reflector
_addInHwndSourceWrapper = contract as AddInHwndSourceWrapper;

if (_addInHwndSourceWrapper != null)
{
    _addInHwndSourceWrapper.RegisterKeyboardInputSite(
        new AddInHostSite(this)); // Internal method call!
}
  
```

To eliminate this error, I created the NativeContractInsulator class. This class lives on the server (plug-in) side. It implements the INativeHandleContract interface by forwarding all calls to the original INativeHandleContract returned from the ViewToContractAdapter method. However, unlike the original implementation, it can't be cast to AddInHwndSourceWrapper. Thus, the cast on the client (host) side isn't successful and the forbidden internal method call doesn't occur.

Examining the Plug-In Architecture in More Detail

The Plugin.Load and Plugin.CreateView methods create all necessary moving parts for plug-in integration.

Figure 5 shows the resulting object graph. It's somewhat complicated, but each part is responsible for a particular role. Together, they ensure seamless and robust operation of the host plug-in system.

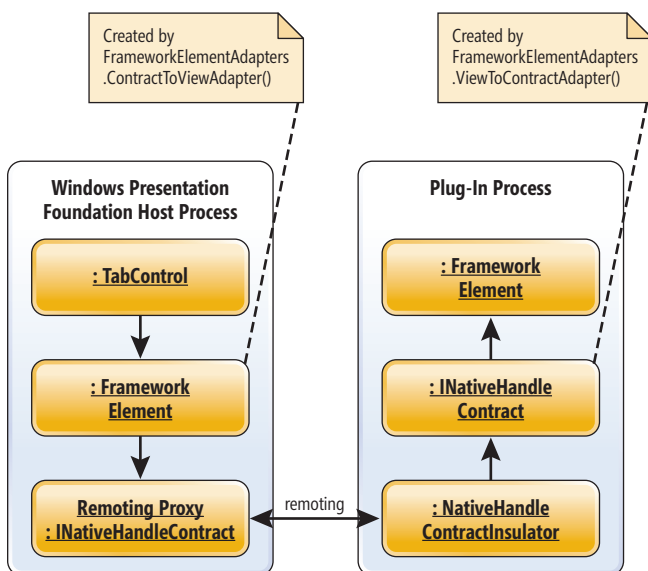


Figure 4 Marshaling a FrameworkElement Between the Plug-In Process and the Host Process

The Plugin class denotes a single plug-in instance in the host. It holds the View property, which is the plug-in's visual representation inside the host process. The Plugin class creates an instance of PluginProcessProxy and retrieves from it an IRemotePlugin. IRemotePlugin contains a remote plug-in control in the form of INativeHandleContract. The Plugin class then takes that contract and converts it to FrameworkElement as shown here (with some code elided for brevity):

```

public interface IRemotePlugin : IServiceProvidor, IDisposable
{
    INativeHandleContract Contract { get; }
}

class Plugin
{
    public void CreateView()
    {
        View = FrameworkElementAdapters.ContractToViewAdapter(
            _remoteProcess.RemotePlugin.Contract);
    }
}
  
```

The PluginProcessProxy class controls the plug-in process lifecycle from within the host. It's responsible for starting the plug-in process, creating a remoting channel and monitoring plug-in process health. It also engages the PluginLoader service and from that retrieves an IRemotePlugin.

The PluginLoader class runs inside the plug-in process and implements the plug-in process lifecycle. It establishes a remoting channel, starts a WPF message dispatcher, loads a user plug-in, creates a RemotePlugin instance and hands this instance to the PluginProcessProxy on the host side.

The RemotePlugin class makes the user plug-in control marshalable across process boundaries. It converts the user's FrameworkElement to INativeHandleContract and then wraps this contract with a NativeHandleContractInsulator to work around illegal method call issues described earlier.

Finally, the user's plug-in class implements the IPlugin interface. Its main job is to create a plug-in control inside the plug-in process. Typically this will be a WPF UserControl, but it can be any FrameworkElement.

When a plug-in load is requested, the PluginProcessProxy class spawns a new child process. The child process executable is either PluginProcess.exe or PluginProcess64.exe, depending on whether the plug-in is 32-bit or 64-bit. Each plug-in process receives a unique GUID in the command line, as well as the plug-in base directory:

```

PluginProcess.exe
PluginProcess.0DAA530F-DCE4-4351-8D0F-36B0E334FF18
c:\plug-in\assembly.dll
  
```

The plug-in process sets up a remoting service of type IPluginLoader and raises a named "ready" event, in this case, PluginProcess.0DAA530F-DCE4-4351-8D0F-36B0E334FF18.Ready. The host then can use IPluginLoader methods to load the plug-in.

An alternative solution would be to have the plug-in process call into the host once it's ready. This would eliminate the need for the ready event, but it would make error handling much more complicated. If the "load plug-in" operation originates from the plug-in process, error information is also retained in the plug-in process. If something goes wrong, the host might never find out about it. Therefore, I chose the design with the ready event.

Another design issue was whether to accommodate plug-ins not deployed under the WPF host directory. On one hand, in the .NET Framework, loading assemblies not located inside the application directory causes certain difficulties. On the other hand, I recognize

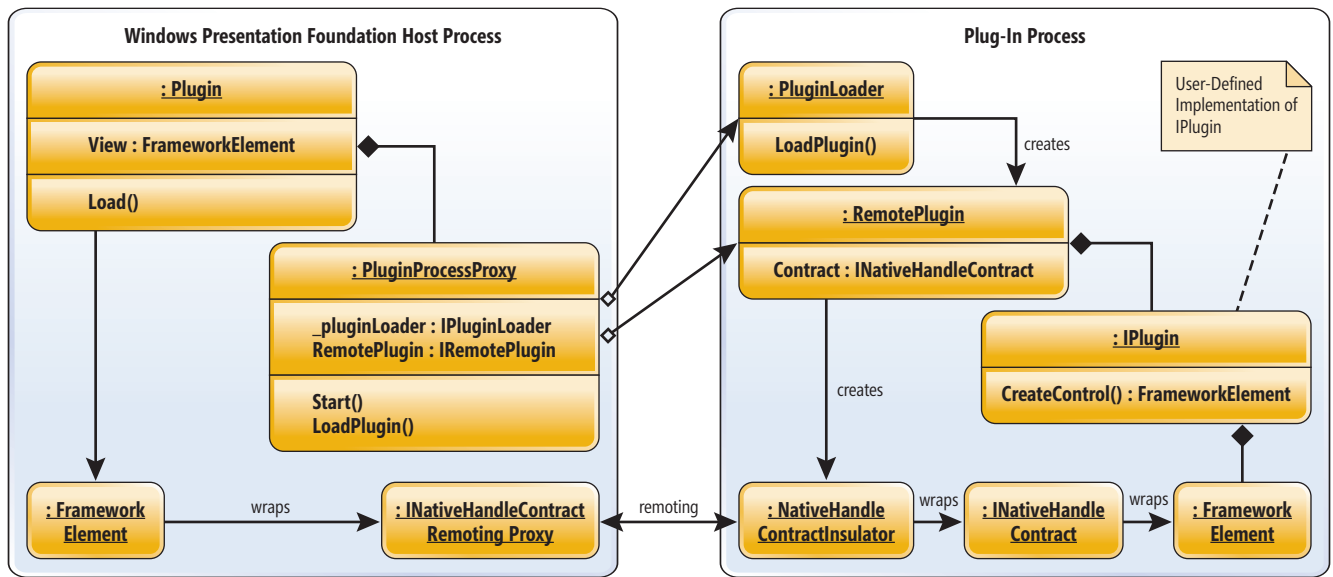


Figure 5 Object Diagram of a Loaded Plug-In

that plug-ins might have their own deployment concerns, and it might not always be possible to deploy a plug-in under the WPF host directory. Moreover, some complex applications don't behave properly when they aren't run from their base directories.

Because of these concerns, the WPF host allows loading plug-ins from anywhere on the local file system. To achieve that, the plug-in process performs virtually all operations in a secondary AppDomain whose application base directory is set to the plug-in's base directory. This creates the problem of loading WPF host assemblies in that AppDomain. This could be achieved in at least four ways:

- Put WPF host assemblies in the Global Assembly Cache (GAC).
- Use assembly redirects in the app.config file of the plug-in process.
- Load WPF host assemblies using one of the LoadFrom/CreateInstanceFrom overrides.
- Use the unmanaged hosting API to start the CLR in the plug-in process with the desired configuration.

Each of these solutions has pros and cons. Putting WPF host assemblies in the GAC requires administrative access. While the GAC is a clean solution, requiring administrative rights for installation can be a big headache in a corporate environment, so I

tried to avoid that. Assembly redirects are also attractive, but configuration files will then depend on the location of the WPF host. This makes an xcopy install impossible. Creating an unmanaged hosting project seemed to be a big maintenance risk.

So I went with the LoadFrom approach. The big downside of this approach is WPF host assemblies end up in the LoadFrom context (see the blog post, "Choosing a Binding Context," by Suzanne Cook at bit.ly/cZmVuz). To avoid any binding issues, I needed to override the AssemblyResolve event in the plug-in AppDomain, so the plug-in's code can find WPF host assemblies more easily.

Developing Plug-Ins

You can implement a plug-in as a class library (DLL) or an executable (EXE). In the DLL scenario, the steps are as follows:

1. Create a new class library project.
2. Reference the WPF assemblies PresentationCore, PresentationFramework, System.Xaml and WindowsBase.
3. Add a reference to the WpfHost.Interfaces assembly. Make sure "copy local" is set to false.
4. Create a new WPF user control, such as MainUserControl.
5. Create a class named Plugin that derives from IKriv.WpfHost.Interfaces.PluginBase.
6. Add an entry for your plug-in to the plugins.xml file of the host.
7. Compile your plug-in and run the host.

A minimal plug-in class looks like this:

```
public class Plugin : PluginBase
{
    public override FrameworkElement CreateControl()
    {
        return new MainUserControl();
    }
}
```

Alternatively, a plug-in can be implemented as an executable. In this case, the steps are:

1. Create a WPF application.
2. Create a WPF user control, for example, MainUserControl.

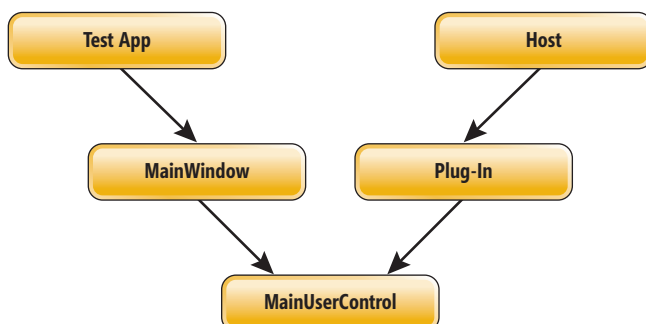


Figure 6 The Class Diagram for a Dual-Head Plug-In

facebook



Microsoft
SharePoint 2010



Linked in



twitter

SEE THE WORLD AS A DATABASE

ADO.NET ▪ JDBC ▪ ODBC ▪ SQL SSIS ▪ ODATA
MYSQL ▪ EXCEL ▪ POWERSHELL



Microsoft Visual Studio Java ODBC Microsoft SQL Server Microsoft Excel Microsoft BizTalk MySQL OData

Work With Relational Data, Not Complex APIs or Services

Whether you are a developer using ADO.NET, JDBC, OData, or MySQL, or a systems integrator working with SQL Server or Biztalk, or even an information worker familiar with ODBC or Excel – our products give you bi-directional access to live data through easy-to-use technologies that you are already familiar with. If you can connect to a database, then you will already know how to connect to Salesforce, SAP, SharePoint, Dynamics CRM, Google Apps, QuickBooks, and much more!



Give RSSBus a try today and see what mean:

visit us online at www.rssbus.com to learn more or download a free trial.

rssbus

INTEGRATION YOUR WAY

3. Add `MainUserControl` to the application's main window.
4. Add a reference to the `WpfHost.Interfaces` assembly. Make sure "copy local" is set to false.
5. Create a class named `Plugin` that derives from `IKriv.WpfHost.Interfaces.PluginBase`.
6. Add an entry of your plug-in to the `plugins.xml` file of the host.

Your plug-in class would look exactly like the preceding example, and your main window XAML should contain nothing but a reference to `MainUserControl`:

```
<Window x:Class="MyPlugin.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:MyProject"
        Title="My Plugin" Height="600" Width="766" >
    <Grid>
        <local:MainUserControl />
    </Grid>
</Window>
```

A plug-in implemented like this can run as a standalone application or within the host. This simplifies debugging plug-in code not related to host integration. The class diagram for such a "dual-head" plug-in is shown in **Figure 6**.

This technique also provides an avenue for quick conversion of

Because the plug-in isn't an independent application, but instead is launched by the host, debugging might not be straightforward.

existing applications to plug-ins. The only thing you need to do is convert the application's main window into a user control. Then instantiate that user control in a plug-in class as demonstrated earlier. The `SolarSystem` plug-in in the accompanying code download is an example of such conversion. The whole conversion process took less than an hour.

Because the plug-in isn't an independent application, but instead is launched by the host, debugging might not be straightforward. You can start debugging the host, but Visual Studio can't yet attach the debugger to the plug-in process once it's running or have the plug-in process break into the debugger on startup by changing line 4 of the `PluginProcess.app.config` to:

```
<add key="BreakIntoDebugger" value="True" />
```

Another alternative is to create your plug-in as a standalone application as described earlier. You can then debug most of the plug-in as a standalone application, only periodically checking that integration with the WPF host works properly.

If the plug-in process breaks into the debugger on startup, you'll want to increase the ready event timeout by changing line 4 of the `WpfHost.app.config` file, as follows:

```
<add key="PluginProcess.ReadyTimeoutMs" value="500000" />
```

A list of example plug-ins available in the accompanying code download and descriptions of what they do is shown in **Figure 7**.

Host Services and Plug-In Services

In the real world, plug-ins often need to use services provided by the host. I demonstrate this scenario in the `UseLogService` plug-in in the code download. A plug-in class might have a default constructor or a constructor that takes one parameter of type `IWpfHost`. In the latter case, the plug-in loader will pass an instance of WPF host to the plug-in. Interface `IWpfHost` is defined as follows:

```
public interface IWpfHost : IServiceProvider
{
    void ReportFatalError(string userMessage, string fullExceptionText);
    int HostProcessId { get; }
}
```

I use the `IServiceProvider` part in my plug-in. `IServiceProvider` is a standard .NET Framework interface defined in `mscorlib.dll`:

```
public interface IServiceProvider
{
    object GetService(Type serviceType);
}
```

I'll use it in my plug-in to obtain the `ILog` service from the host:

```
class Plugin : PluginBase
{
    private readonly ILog _log;
    private MainUserControl _control;

    public Plugin(IWpfHost host)
    {
        _log = host.GetService<ILog>();
    }

    public override FrameworkElement CreateControl()
    {
        return new MainUserControl { Log = _log };
    }
}
```

The control can then use the `ILog` host service to write to the host's log file.

The host can also use services provided by plug-ins. I defined one such service called `IUnsavedData`, which proved to be useful in real life. By implementing this interface, a plug-in may define a list of unsaved work items. If the plug-in or the whole host application is closed, the host will ask the user whether he wants to abandon unsaved data, as shown in **Figure 8**.

The `IUnsavedData` interface is defined as follows:

```
public interface IUnsavedData
{
    string[] GetNamesOfUnsavedItems();
}
```

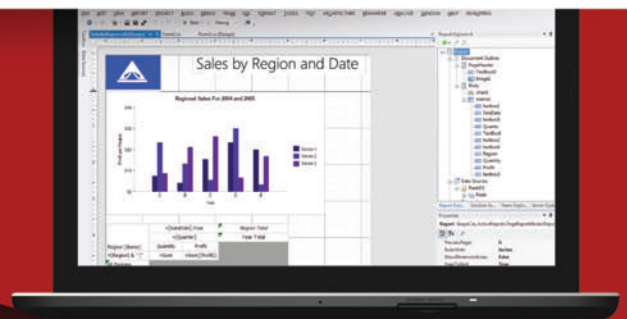
A plug-in author doesn't need to implement the `IServiceProvider` interface explicitly. It's enough to implement the `IUnsavedData` interface in the plug-in. The `PluginBase.GetService` method will

Figure 7 Example Plug-Ins Available in the Accompanying Code Download

Plug-In Project	What It Does
BitnessCheck	Demonstrates how a plug-in can run as 32-bit or 64-bit
SolarSystem	Demonstrates an old WPF demo application converted to a plug-in
TestExceptions	Demonstrates exception handling for user thread and worker thread exceptions
UseLogServices	Demonstrates use of host services and plug-in services

REPORTS EVERYWHERE

We are the Reporting experts and we care about performance and Visual Studio integration as much as you. The all new ActiveReports 8 reporting engine delivers a granular API that allows you to create concise and beautiful reports. Whether you need multiple outputs, the ability to create maps, barcodes, or the power of HTML5 to display your reports, ActiveReports 8 can handle all of your heavy reporting requirements.



Sophisticated, Fast
& Powerful Reports

ALL NEW ACTIVEREPORTS 8

Flexible Layouts Using
Section, Region & Fixed
Page Designers

HTML5 & Touch Optimized
Report Viewers for Mobile
Devices

A New Scalable, Distributed
& Load Balanced Enterprise-
grade Report Server

take care of returning it to the host. My UseLogService project in the code download provides a sample IUnsavedData implementation, with relevant code shown here:

```
class Plugin : PluginBase, IUnsavedData
{
    private MainUserControl _control;

    public string[] GetNamesOfUnsavedItems()
    {
        if (_control == null) return null;
        return _control.GetNamesOfUnsavedItems();
    }
}
```

Logging and Error Handling

WPF host and plug-in processes create logs in the %TMP%\WpfHost directory. The WPF host writes to WpfHost.log and each plug-in host process writes to PluginProcess.Guid.log (the “Guid” isn’t part of the literal name, but is expanded to the actual Guid value). The log service is custom-built. I avoided using popular logging services such as log4net or NLog to make the sample self-contained.

A plug-in process also writes results to its console window, which you can show by changing line 3 of the WpfHost app.config to:

```
<add key="PluginProcess.ShowConsole" value="True" />
```

I took great care to report all errors to the host and handle them gracefully. The host monitors plug-in processes and will close the plug-in window if a plug-in process dies. Similarly, a plug-in process monitors its host and will close if the host dies. All errors are logged, so examining log files helps tremendously with troubleshooting.

It’s important to remember that everything passed between the host and the plug-ins must be either [Serializable] or of type derived from MarshalByRefObject. Otherwise, .NET remoting won’t be able to marshal the object between the parties. The types and interfaces must be known to both parties, so typically only built-in types and types from WpfHost.Interfaces or PluginHosting assemblies are safe for marshaling.

Generally, you should regard
plug-ins as third-party code not
under control of the host authors.

Versioning

WpfHost.exe, PluginProcess.exe and PluginHosting.dll are tightly coupled and should be released simultaneously. Fortunately, plug-in code doesn’t depend on any of these three assemblies and therefore they can be modified in almost any way. For example, you can easily change the synchronization mechanism or the name of the ready event without affecting the plug-ins.

The WpfHost.Interfaces.dll component must be versioned with extreme care. It should be referenced, but not included with the plug-in code (CopyLocal=false), so the binary for this assembly always comes only from the host. I didn’t give this assembly a strong name because I specifically don’t want side-by-side execution. Only one version of WpfHost.Interfaces.dll should be present in the entire system.

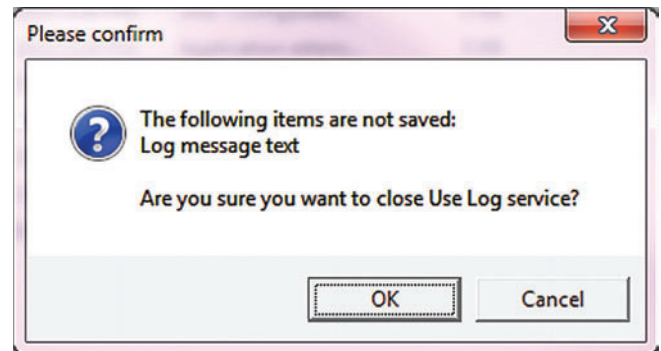


Figure 8 Using the IUnsavedData Service

Generally, you should regard plug-ins as third-party code not under control of the host authors. Modifying or even recompiling all plug-ins at once might be difficult or impossible. Therefore, new versions of the interface assembly must be binary-compatible with previous versions, with the number of breaking changes held to an absolute minimum.

Adding new types and interfaces to the assembly is generally safe. Any other modifications, including adding new methods to interfaces or new values to enums, can potentially break binary compatibility and should be avoided.

Even though the hosting assemblies don’t have strong names, it’s important to increment the version numbers after *any* change, however small, so no two assemblies with the same version number have different code.

A Good Starting Point

My reference architecture provided here isn’t a production-quality framework for plug-in-host integration, but it comes quite close and can serve as a valuable starting point for your application.

The architecture takes care of boilerplate yet difficult considerations such as the plug-in process lifecycle, marshaling plug-in controls across processes, a mechanism for exchange, and discovery of services between the host and the plug-ins, among others. Most design solutions and workarounds aren’t arbitrary. They’re based on actual experience in building composite applications for WPF.

You’ll most likely want to modify the visual appearance of the host, replace the logging mechanism with the standard one used in your enterprise, add new services and possibly change the way plug-ins are discovered. Many other modifications and improvements are possible.

Even if you don’t create composite applications for WPF, you might still enjoy examining this architecture as a demonstration of how powerful and flexible the .NET Framework can be and how you can combine familiar components in an interesting, unexpected and productive way. ■

IVAN KRIVYAKOV is a technical lead at Thomson Reuters. He’s a hands-on developer and architect who specializes in building and improving complex line-of-business (LOB) Windows Presentation Foundation applications.

THANKS to the following Microsoft technical experts for reviewing this article: Dr. James McCaffrey, Daniel Plaisted and Kevin Ransom

SpreadsheetGear

Performance Spreadsheet Components

SpreadsheetGear 2012 Now Available

NEW!

WPF and Silverlight controls, multithreaded recalc, 64 new Excel compatible functions, save to XPS, improved efficiency and performance, Windows 8 support, Windows Server 2012 support, Visual Studio 2012 support and more.

Excel Reporting for ASP.NET, WinForms, WPF and Silverlight



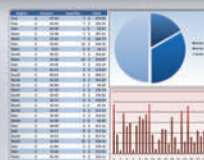
Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.

Excel Compatible Windows Forms, WPF and Silverlight Controls



Add powerful Excel compatible viewing, editing, formatting, calculating, filtering, charting, printing and more to your Windows Forms, WPF and Silverlight applications with the easy to use WorkbookView controls.

Excel Dashboards, Calculations, Charting and More



You and your users can design dashboards, reports, charts, and models in Excel or the SpreadsheetGear Workbook Designer rather than hard to learn developer tools and you can easily deploy them with one line of code.

**Free
30 Day
Trial**

Download our fully functional 30-Day evaluation and bring Excel Reporting, Excel compatible charting, Excel compatible calculations and much more to your ASP.NET, Windows Forms, WPF, Silverlight and other Microsoft .NET Framework solutions.

www.SpreadsheetGear.com



SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

This article is from *MSDN Magazine*'s special coverage of application development in the government sector. Read more articles like this in the Special Government Issue (msdn.microsoft.com/magazine/dn463800) of *MSDN Magazine*.

Create Modern Microfiche with the Chronicling America API

Tim Kulp

Today, the newspaper is being replaced by other news sources on the Web. At one time, though, newspapers were the primary source of information about current events. Through the Chronicling America project (chroniclingamerica.loc.gov), the U.S. Library of Congress and various institutions have been working to make the rich history of U.S. newspapers available on the Web via a standardized API. This API lets developers leverage media from the Chronicling America project to build research apps, news apps with historical context or apps for attaching content to family genealogy. In this article, I'll introduce the Chronicling America API and build a Windows Store app that lets users research and view digitalized newspapers.

Overview of the Chronicling America API

Chronicling America has two core components to its API: the OpenSearch protocol and standardized URLs to navigate content. For each publication, the URL follows a standard format that you can modify to provide different data. As an example, retrieving a specific page requires the following URL structure: <http://chroniclingamerica.loc.gov/lccn/sn83030214/1913-08-12/ed-1/seq-1/>. This format breaks down to: requesting a specific publication ([/lccn/sn83030214/](http://chroniclingamerica.loc.gov/lccn/sn83030214/)), providing the date and identifier of the issue and edition (1913-08-12/ed-1), and providing the page number (seq-1/). If I want to request the second page, I just update seq-1 to seq-2. Throughout this article, I'll use the standardized URL to make

navigation simple and fast without having to look up objects with many one-off requests to the server.

You can use the URL to retrieve specific content, and the OpenSearch protocol to search titles and pages. Searches against titles and pages use different URL structures. Title search structures can be found at 1.usa.gov/1a0G0pF, and page search structures can be found at 1.usa.gov/188T1HB. These XML files provide all the output formats and parameters that can be used in Chronicling America. I'll focus on page searches in this article to have the most properties against which to search.

Building a Microfiche App

If you aren't old enough to remember microfiche, it was a clear piece of film on which miniature images were printed. These images were barely readable by the naked eye, but by putting the microfiche sheet in a special projector, you could gain access to hundreds of images (or more). It was a great mass-storage medium for images and helped users quickly move between documents by using the projector's control wheel.

Microfiche is my inspiration for a sample app that displays newspapers from Chronicling America. The app starts with a list of publications displayed in a GridView. Users can search this content (an enhancement from microfiche) and examine an issue, much like using an eReader. The Library of Congress has already built a nice Web application with robust JavaScript capabilities. You can learn from this site and translate JavaScript from the Web to build the Windows Store app.

Code download available at msdn.microsoft.com/magazine/msdnmag0114.

I built my *Chronicling America* app in JavaScript for Windows 8.1. To begin, create a Navigation app from the JavaScript Windows Store app templates in Visual Studio. Use the preexisting code in the template (for example, the navigation code in *navigator.js* and the app activation event handler in *default.js*) so you don't have to manually build all your app's infrastructure.

100 Years Ago Today ...

The *Chronicling America* homepage displays newspaper front pages for today's date 100 years ago. This is a fun place to start my app and immediately introduce people to the content. To begin, I need to connect to *Chronicling America* and pull data. The app performs many functions around its data, so in the *js* folder, I create a new JavaScript file named *data.js*, where all data interactions will occur. Think of this as my data layer.

For each publication,
the URL follows a standard
format that you can modify to
provide different data.

I add the following code to *data.js* to create a self-contained function, because I don't want the code in *data.js* to leak out into the global namespace:

```
(function () {  
    "use strict";  
    // Code goes here  
  
})();
```

This block makes anything inside it private so I don't encounter any variable-naming conflicts with files that use *data.js*. The comment, "Code goes here" is replaced with the code in **Figure 1**.

Figure 1 Initial Code for Data.js

```
var imageUrl = new WinJS.Binding.List(); // The image list for binding  
var pagesList = new WinJS.Binding.List(); // List of pages for a single issue  
  
var urlPrefix =  
    "http://chroniclingamerica.loc.gov"; // Used to prefix all URLs  
var httpClient =  
    new Windows.Web.Http.HttpClient(); // Default HTTP client to use  
  
function getInitialPages() {  
    // Get the date and set it 100 years ago  
    var hundredYearsAgo = new Date();  
    hundredYearsAgo.setYear(hundredYearsAgo.getYear() - 100);  
  
    var uri =  
        new Windows.Foundation.Uri(urlPrefix + "/frontpages/" +  
            hundredYearsAgo.getFullYear() + "-" + hundredYearsAgo.getMonth() + "-" +  
            hundredYearsAgo.getDate() + ".json");  
  
    httpClient.getStringAsync(uri)  
        .then(  
            function c(result) {  
                var images = JSON.parse(result);  
                images.forEach(function (item) {  
  
                    item.medium_url = urlPrefix + item.medium_url;  
                    item.thumbnail_url = urlPrefix + item.thumbnail_url;  
                    item.url = urlPrefix + item.url;  
  
                    imageUrl.push(item);  
                });  
            })  
        );  
  
    // Get function goes here  
  
    WinJS.Namespace.define("Data", {  
        GetInitialPage: getInitialPages,  
  
        imageUrl: imageUrl,  
        PageList: pagesList  
    });
```

Figure 1 starts with basic declarations that hold the data from the *Chronicling America* API calls. I set up some housekeeping variables, such as a URL prefix to avoid typing (and inevitably mistyping) the base *Chronicling America* URL, as well as an *HttpClient* object to be shared across functions.

The first function in *data.js* calls the initial pages from 100 years ago. This function starts with a simple calculation to get today's date

Debrief: Engaging Experience, Reusable Skills

Windows Store apps let you unleash your creativity with data. In this app, you can explore historical newspaper information that could be used for research, education or other historical inquiries. Through a media-rich API and simple JavaScript coding, combining *Chronicling America* with Windows 8.1 provides an easy way to build a deeply engaging experience.

IT Brief:

In this article, data is pulled from an API and searched, and the content is explored—all by reusing a core set of skills. An engaging experience is created through code reuse and a media-rich API.

- Engage your development team members by using their existing skills to build rich experiences for users.
- Engage your users by connecting to Windows 8.1 charm features such as Search.
- Provide a new vision of an old technology as a market differentiator.

Dev Brief:

Development is a creative exercise, and Windows Store apps give you a lot of freedom. In this article, a retro technology is reimagined and a core set of skills (*ListView* and *Windows.Web.Http.HttpClient*) is reused to build it out. Integrating the app with search features allows users to explore the app's content from anywhere in Windows 8.1.

- Chaining promises to build asynchronous code workflows.
- Reusing *ListView* skills for *FlipView* controls.
- Using the JavaScript prototype capabilities to dynamically extend objects.

More Information:

- Asynchronous programming in JavaScript (Windows Store apps): bit.ly/17EiJtJ
- *Chronicling America*: chroniclingamerica.loc.gov
- *OpenSearch*: opensearch.org

Figure 2 Code Added to the Ready Function

```
ready: function (element, options) {
    // Load the data
    Data.GetInitialPage();

    // Configure the list view
    var hundredYearListView =
        document.getElementById("hundredsYearsListView").winControl;
    hundredYearListView.itemDataSource = Data.ImageList.dataSource;
    hundredYearListView.itemTemplate = document.querySelector(".itemtemplate");
    hundredYearListView.oniteminvoked = this._itemInvoked;

    // Process the bindings
    WinJS.Binding.processAll();
},

_itemInvoked: function (args) {
    args.detail.itemPromise.done(function itemInvoked(item) {
        var itemSelected = Data.ImageList.getAt(args.detail.itemIndex);
        WinJS.Navigation.navigate(
            "/pages/details/details.html", { selectedItem: itemSelected });
    });
}
```

Figure 3 Styles for Home.css

```
.homepage section[role=main] {
    margin-left: 120px;
    margin-right: 120px;
}

.homepage item {
    height: 225px;
    display: block;
    margin-bottom: 5px;
    margin-top: 5px;
    margin-left: 5px;
}

#hundredYearsListView {
    height: 90%;
}
```

and then subtracts 100 years. Using the standardized URLs from Chroniching America, I set up a request to the frontpages directory of the API to load all the front pages from every issue matching the provided date. Notice the URL ends with .json to denote that I want JSON data back from the server.

The Chroniching America API lets you set this value to request different data types, such as XML or HTML. Using the `httpClient.getStringAsync` function, the app requests the JSON data from the API and parses the returned string value into a JSON object. Because I'm calling `getStringAsync`, I don't know what type of data is coming back from the server. My app expects JSON, so to verify the data the API returns can be converted to JSON, I use `JSON.parse`. This returns an array of JSON objects in this case. If the returned string can't be converted to JSON, the code raises an error. Finally, the code updates the URLs in the item to append the prefix (because

Chroniching America provides the URLs here as relative URLs, not absolute) to ensure my app can connect to the images.

One note on using `Windows.Web.Http.HttpClient`: Always plan on failure. The server your code is trying to reach could be down or unavailable, for example. Error handling has been omitted here for brevity, but I've included it in the online sample code.

At the end of the code in **Figure 1**, I added `WinJS.Namespace` to register the contents of `data.js` to be available to other applications. This namespace capability converts the private objects in `data.js` to public, accessible by other code. Using `WinJS.Namespace.define` informs an app that the defined object and members are available to other parts of the code. In essence, this is where you make `Data` and its members accessible with IntelliSense on your pages.

One note on using
`Windows.Web.Http.HttpClient`:
Always plan on failure.

Binding to the Home Screen With the `Data` object defined, you need to connect the HTML page to display content from the Chroniching America site. Start by opening the default.html page and add a script reference to `data.js`.

This step makes `data.js` available to all the pages in the app. Replace the code in the Main section of `home.html` with the following:

```
<div id="hundredsYearsListView" class="itemslist win-selectionstylefilled"
    aria-label="List of Publications"
    data-win-control="WinJS.UI.ListView" data-win-options=
        "{ selectionMode: 'none', layout: { type: WinJS.UI.GridLayout } }"></div>
```

This code defines a `ListView` control that displays search results from the Chroniching America API. The `ListView` control uses a template with an image of the newspaper page and the place of publication. By using a `WinJS.Binding.Template` control, you describe what should appear in the `ListView`. Templates let your design team (or marketing team, if you don't have a design team)



Figure 4 Chroniching America Home Screen

Figure 5 Get Function for Data.js

```
function get(url) {
    pagesList.length = 0;
    var uri = new Windows.Foundation.Uri(
        url.replace(/\/seq-[1-999][1,3](\\|\/\\.json)?$/g, ".json"));
    httpClient.getStringAsync(uri)
        .then(
            function complete(result) {
                var editionItem = JSON.parse(result);
                editionItem.pages.forEach(function (item) {
                    item.thumbnail_url = item.url.replace(".json", "/thumbnail.jpg");
                    item.medium_url = item.url.replace(".json", "/medium.jpg");
                    pagesList.push(item);
                })
            })
}
```

Figure 6 Code for Details.html

```
<div class="listview-itemtemplate" data-win-control="WinJS.Binding.Template">
    <div class="item">
        
    </div>
</div>

<div class="details fragment">
    <header aria-label="Header content" role="banner">
        <button data-win-control="WinJS.UI.BackButton"></button>
        <h1 class="titlearea win-type-ellipsis">
            <span class="pagetitle"></span>
        </h1>
    </header>
    <section aria-label="Main content" role="main">
        <div id="flipview" data-win-control="WinJS.UI.FlipView"></div>
    </section>
</div>
```

work directly with HTML without worrying about data binding. Add the following directly below the `<body>` tag:

```
<div class="itemtemplate" data-win-control="WinJS.Binding.Template">
    <div class="item">
        
        <div class="item-description">
            <h3 class="item-title" data-win-bind="textContent: label"></h3>
            <h4><span data-win-bind="textContent: pages"></span> pp.</h4>
            <h4 class="item-subtitle win-type-ellipsis" data-win-bind="textContent:
                place_of_publication"></h4>
        </div>
    </div>
</div>
```

Determining which property to bind to which field is handled by the `data-win-bind` attribute, which Windows uses to determine what to display in specific attributes during binding. The dataset attribute is a new attribute in HTML5 that lets you build custom attributes for your needs. Windows Store apps make extensive use of the `data-` attribute, but you can build your own attributes, such as `data-publication` in this example, which holds the name of the newspaper. This attribute can be searched by using the `document.querySelector` function, providing easy access to all the elements with this attribute.

Connecting Data to the ListView To put data into the ListView, open the `home.js` file (`/pages/home/home.js`) and add the code shown in **Figure 2** to the `ready` function for the Home page control.

The code in **Figure 2** begins the process of loading data from the API by calling the `Data.GetInitialPage` function. Then I set up the ListView to pick up data from the `ImageList` (which is populated by the `GetInitialPage` function) and associate the template control to the ListView control. To wrap up the ListView controls capabilities, I add

an event handler for `ItemInvoked` so when a user clicks a newspaper image, something happens. The `ready` function ends with a call to `WinJS.Binding.processAll` to update the UI based on the modifications that happened during the `ready` function, such as data binding.

Next, `_itemInvoked` is defined to take a selected item and pass it on to the details page for further action. This is a basic operation of just passing the data object directly to the page. Keep in mind the `navigate` method provides great flexibility for passing data from one page to another. For more information on using navigation in Windows Store apps, check out the samples on the Windows Dev Center Windows Store apps page (bit.ly/153fDXM).

Before you run your app, add the styles shown in **Figure 3** to `home.css` (`/pages/home/home.css`) to give the app a slightly cleaner look.

Run your project at this point, and you should see something that looks like **Figure 4**.

I've built a home screen and a bit of plumbing. The next step is to view the pages of a publication and create a details screen.

Viewing Specific Pages

Adding a details screen is simple with the existing app setup. First, I need to add a new function to `data.js` to look up the pages of an issue. Next, I need to build out the page control that will actually display this data.

Start by going back to `data.js` and replacing the comment, "Get function goes here," with the code in **Figure 5**.

This code takes a URL and uses standard URL conventions of *Chronicling America* to replace the reference to page one (`/seq-1/`) with a JSON call to the edition (converting `ed-1/seq-1/` to `ed-1.json`). To do this, use a regular expression that matches the pattern `/seq-[1-3 digit number]/` or `seq-[1-3 digit number].json` and replace it with just `.json`. By calling the edition's JSON object, I get information about the issue and not just a specific page. I could go further up the URL to get more general information about the issues or publication in general, but for now

☞ The Bemidji daily pioneer.



Figure 7 Sample Details Page

Figure 8 Search Function

```
function search(terms) {
    itemList.length = 0; // Clear the list items
    var uri = new Windows.Foundation.Uri(
        urlPrefix + "/search/pages/results?andtext=" + terms + "&format=json");
    return httpClient.getStringAsync(uri)
        .then(
            function complete(result) {
                var searchList = JSON.parse(result);
                itemList.pop();
                searchList.items.forEach(function (item) {
                    item.medium_url = urlPrefix + item.id + "medium.jpg";
                    item.thumbnail_url = urlPrefix + item.id + "thumbnail.jpg";
                    item.label = item.title;
                    itemList.push(item);
                })
            })
}
```

I'll stick to the edition and pages. After adding the Get function, be sure to register it in the WinJS.Namespace.define function as Get so that other pages can use it for reference.

Creating a Details Page First, under the pages directory, create a folder named "details." This folder will hold the three files that make up a Page control (the HTML, JS and CSS files). Add a new item to the folder, select Page Control (under Windows Store), and name the item details.html. Now update the body tag of details.html (/pages/details/details.html) by using the code in **Figure 6**.

Notice this code is quite similar to home.html, except I use a FlipView control instead of a ListView control. This gives the app more of the feeling of an eReader. Although I'm discussing only images, Chronicling America provides access to PDF content, Optical Character Recognition (OCR) text and other types of content for consumption. Think about how these different types could add more depth to an app (such as assistive technologies that rely on OCR capabilities) for a future project.

The HTML is complete. Now head over to the CSS file (/pages/details/details.css) to add the cleanup styles that will make the page look decent:

```
.details section[role=main] {
    margin-left: 120px;
    margin-right: 120px;
}

#pageListView {
    height: 90%;
    width: 90%;
}

#Flipview {
    height: 90%;
    width: 90%;
}
```

Figure 9 Updating SearchResults.html

```
<div class="itemtemplate" data-win-control="WinJS.Binding.Template">
  <div class="item">
    
    <div class="item-content">
      <h3 class="item-title win-type-x-small win-type-ellipsis"
        data-win-bind="innerHTML: title searchResults.markText"></h3>
      <h4 class="item-subtitle win-type-x-small win-type-ellipsis"
        data-win-bind="innerHTML: publisher searchResults.markText"></h4>
      <h4 class="item-description win-type-x-small win-type-ellipsis"
        data-win-bind="innerHTML:
          place_of_publication searchResults.markText"></h4>
    </div>
  </div>
</div>
```

Connecting Details to the FlipView

Open the details.js page (/pages/details/details.js) and add the following code to the ready function of the page control:

```
// Step 1: consume the options data
var selectedItem = options.selectedItem;

// Step 2: acquire data for pages
Data.Get(selectedItem.url);

// Step 3: set the page title
var title = document.querySelector(".details .pagetitle");
title.textContent = selectedItem.label;

// Step 4: set up the ListView to display the pages
var fV = document.getElementById("flipview").winControl;
fV.itemDataSource = Data.PageList.dataSource;
fV.itemTemplate = document.querySelector(".flipview-itemtemplate");

WinJS.Binding.processAll();
```

Much of this code is the same as in home.js, with the exception that I use options.selectedItem. This object comes from the homepage and is passed into the details page through the navigate function. I use data from the options.selectedItem object to call Data.Get and populate the pageList object for binding to the FlipView control. Just like the ListView, the FlipView uses a WinJS.Binding.Template control to display content. I decided to use a FlipView because I want to display only one page at a time, and that page is a picture. Using the controls for a FlipView also provides a more natural reading experience to users.

Run the code, and you should see something like **Figure 7**.

With the details in place, there's one more piece to wrap up for the app: search capability.

Searching Content

The ability to search newspaper content provides a great research tool. It's a definite improvement from the microfiche devices of old. By using OpenSearch, Chronicling America provides a powerful search capability that follows an open standard. For my app, I'm going to search against page content using the parameters from 1.usa.gov/188T1HB.

Begin by adding a folder in the pages directory named "search." Add a new item to the search folder and select Search Results control. To enable search, you must add the Search declaration to your appmanifest and a reference to the searchResults.js file to the default.html file. There's more to search than adding a few files, so if you aren't familiar with the Search contract process, you can get started by reading the Windows Store apps Dev Center article at bit.ly/HMtNCo.

The first element needed to build the search capability is to add a new Search function to the data.js (/js/data.js) file. Open data.js and add the code in **Figure 8** after the get function.

This simple function takes search terms and builds out the URI to work with the page search URL. The andtext parameter lets you provide search terms to use for content on the page, but remember to check the search definition XML file for more parameters that could be used. To complete data.js, add Search as an item in the namespace so that searchResults.js (/pages/search/searchResults.js) can access it.

Modifying the Search Results Control When you add the item, three files are created (just as with a page control): an HTML file for displaying your results, a CSS file for styling your results and a JavaScript file for binding your results. Update the searchResults.html file first by replacing the itemtemplate to match the data model, as shown in **Figure 9**.

Figure 10 Code for `_searchData`

```
_searchData: function (queryText) {
    var results;
    if (window.Data) {
        Data.Search(queryText).done(
            function complete(result) {
                if (Data.ImageList.length > 0)
                    document.querySelector('.resultsmessage').style.display = "none";
            });
        results = Data.ImageList.createFiltered(function (item) {
            return ((item.title !== undefined && item.title.indexOf(queryText) >= 0)
                || (item.publisher !== undefined && item.publisher.indexOf(queryText)
                    >= 0) || (item.place_of_publication !== undefined &&
                    item.place_of_publication.indexOf(queryText) >= 0));
        });
    } else {
        results = new WinJS.Binding.List();
    }
    return results;
}
```

Now open the `searchResults.js` file. This file has a number of functions in it, but most of the changes you make are to the `_searchData` function. Replace the code for `_searchData` with the code shown in **Figure 10**.

If you compare this code to the code that was generated, you'll see all I did was update the item object's data model to reflect the components and add some functionality to occur after the `Data.Search` function completes. Using the pre-generated code, you can get an app like this to market more quickly. You can focus on creating the specific features of your apps and not have to separately hook up each piece every time.

Before you run the code, update the filter `_generateFilters` function in `searchResults.js` to reflect the data model. By default, two filters are added for Groups. Update those to be something more useful for the data model such as a filter on publication city

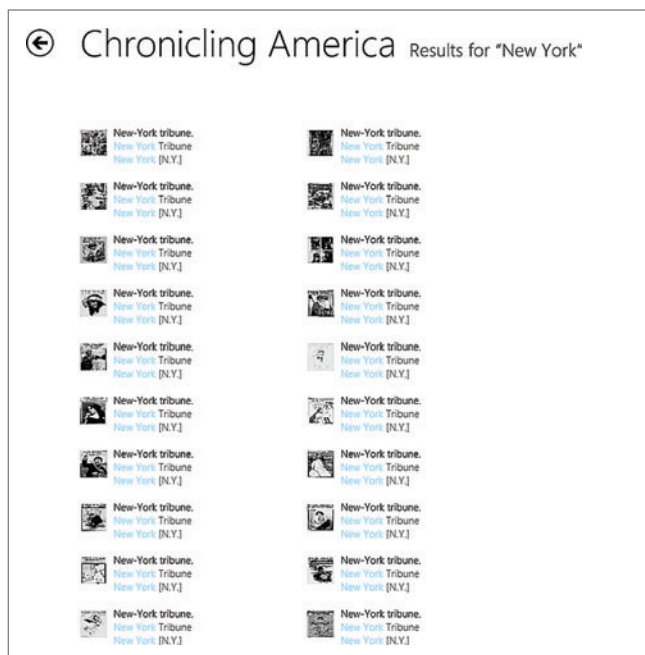


Figure 11 The Search Results Screen for the Chronicling America App

values. With this work complete, you should be able to run your code and see something similar to **Figure 11**.

To wrap up the search screen, you need to view the details of an article to see its pages. Update the `_itemInvoked` function in `searchResults.js` (`/pages/search/searchResults.js`) with the following code:

```
_itemInvoked: function (args) {
    args.detail.itemPromise.done(function itemInvoked(item) {
        var searchResult = originalResults.getAt(args.detail.itemIndex);
        WinJS.Navigation.navigate(
            "/pages/details/details.html",
            { selectedItem: searchResult });
    });
},
```

Using the pre-generated code, you can get an app like this to market more quickly. You can focus on creating the specific features of your apps and not have to separately hook up each piece every time.

This code captures the currently selected search result item and passes it to the details page. The `WinJS.Navigation.navigate` function can take two arguments: the address of the page and an initial state declaration. By using the `initialState` object, you can pass parameters from one screen to the next. In this case, the `navigate` function passes the selected `searchResult` object to the details page just like I used the `selectedItem` from the homepage.

Microfiche for the 21st Century

Chronicling America archives historical newspapers and makes them available through a simple API. In this article, I explored the core capabilities of this Web service and how to consume it in a Windows Store app by using some basic controls. Windows 8.1 provides many opportunities for you to be creative and rethink how people interact with technology.

In this case, I took what would've been a sheet of newspaper images and made it into an interactive modern app. You could extend this example to share and store the user's personalized search settings based on his interests—and this is just the tip of the iceberg. The Library of Congress and National Endowment for the Humanities have provided a rich API on which to build. You can add this historical data to your cookbook for building great apps. ■

TIM KULP leads the development team at UnitedHealthcare International in Baltimore. You can find Kulp on his blog at seccode.blogspot.com or on Twitter at twitter.com/seccode, where he talks code, security and the Baltimore foodie scene.

THANKS to the following technical expert for reviewing this article:
Eric Schmidt (Microsoft)

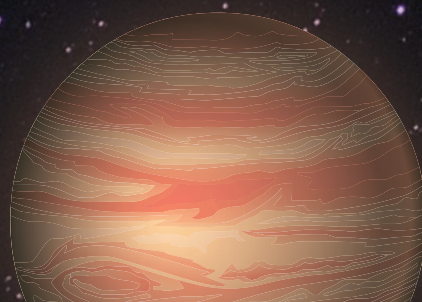
**LIVE!****LIVE!****LIVE!****LIVE!****LIVE!**

Visual Studio SQL Server SharePoint Web Dev Modern Apps

COMPREHENSIVE TRAINING



The Developer World is always changing; new technologies emerge, current ones evolve and demands on your time grow. Live! 360 DEV offers comprehensive training through 5 co-located events on the most relevant and leading edge technologies in your world today. You'll learn from pre-eminent experts in the industry, network with like-minded peers, and return home with the knowledge and solutions you need to tackle your biggest development challenges.



Live! 360 DEV Explores:

- **Visual Studio Live!** – May the Code be with you. The most trusted source in the universe for .NET Training for 21 years and counting.
- **Web Dev Live!** – NEW EVENT! Web Dev Live! will dive deep to explore all that is HTML5, JavaScript and ASP.NET.
- **Modern Apps Live!** – Launch your mobile, cross-device & cloud development training here.
- **SharePoint Live!** – Set your course for collaboration with these sessions designed strictly for devs.
- **SQL Server Live!** – Your Mission? Conquering coding against SQL Server.
- This means you have five events with over a hundred sessions to choose from – mix and match sessions to create your own, custom event line-up - it's like no other dev conference available today.

TURN THE PAGE FOR MORE EVENT DETAILS →

FOR THE DEVELOPER WORLD



Explore the World of Live! 360 DEV



Visual Studio

Visual Studio Live!

Much like outer space, the .NET development platform is ever-changing and constantly expanding. Visual Studio Live! exists to help guide you through this universe, featuring code-filled days, networking nights and independent education. Whether you are a .NET developer, software architect or a designer, Visual Studio Live!'s multi-track events include focused, cutting-edge education on the .NET platform that you'll be ready to implement as soon as you get back to the office.



Modern Apps

Presented in
Partnership with

Magenic

Modern Apps Live!

Presented in partnership with Magenic, Modern Apps Live! brings development managers, software architects, and development leads together to break down the latest and greatest techniques in low-cost, high-value application development. What sets Modern Apps Live! apart is the singular topic focus; sessions build on each other as the conference progresses, leaving you with a holistic understanding of modern applications.

And INTRODUCING...



Web Dev

Web Dev Live!

The producers of Live! 360 DEV bring you a new, dynamic, content-rich event — Web Dev Live! We're seeing an evolution where server-side web technologies continue to improve, and at the same time JavaScript and HTML 5 are becoming viable smart client development technologies that operate on their own. This is a great time to jump into the JavaScript programming world, or to update your knowledge of web development based on the latest tools and capabilities.



SharePoint

SharePoint Live!

This March, SharePoint Live! will be for developers only, where expert speakers will guide an exploration of patterns and practices, highlighting the capabilities of various development tools. Select sessions will focus on methodologies for development that will supplement your overall Live! 360 DEV experience.



SQL Server

SQL Server Live!

Developers have an incredible amount of impact on the overall performance of a database application, and SQL Server Live! shines a spotlight to help developers deliver more. From T-SQL enhancements to "never do this" worst-practices reviews, there's plenty of content to supplement your Live! 360 DEV agenda!



Scan the QR code to register or for more event details.



CONNECT WITH LIVE! 360



twitter.com/live360events



facebook.com/live360events



Join the "Live! 360" Group

REGISTER BY JANUARY 22 AND SAVE \$400

Use promo code DEVJAN4



Troubleshooting Applications with IIS Logs

Eduardo Sanabria

Have you ever tried to troubleshoot or debug an application without having ever seen the code? Have you ever had a malfunctioning application, and neither the browser nor the application provided a useful error code?

I've encountered both scenarios many times, and it's a good idea to be prepared for their eventuality. The techniques I'll describe in this article will help you troubleshoot any application or system running under IIS, no matter what platform it's coded on. These techniques have helped me troubleshoot applications and Web sites in a variety of situations, especially on devices other than PCs—a scenario that is becoming the norm these days. In a recent case, these techniques helped me discover why videos wouldn't display on Apple devices though they ran perfectly on Windows-based devices.

Some Considerations

There are many valid techniques for debugging ASP.NET and other Web applications running under IIS. The browser itself often produces a specific error or set of errors that is enough to resolve the issues.

This article discusses:

- Turning on IIS logging
- Finding a Web site's ID number
- Understanding the contents of log files

Technologies discussed:

Microsoft .NET Framework, Windows Server, IIS

But suppose that information isn't enough? This is where it becomes handy to know a few extra techniques. The simplest of these is also one of the quickest and best-known—running the application directly on the server. Sometimes the servers aren't configured for this option, but if you can do it, the server will generally provide more useful debug information than an outside machine. This behavior is obviously built-in by Microsoft for security purposes. To get even more data on a server's browser, turn off the "Show friendly HTTP error messages" option, which you'll find in Internet Explorer, under Internet Options | Advanced.

Sometimes you'll need more information, and that's when logging becomes essential. Microsoft has built powerful logging facilities into its servers, which will help immensely on any troubleshooting quest, assuming you know what to look for, and where.

Turning on IIS Logging

The first step is to turn on Windows logging on the server. There are several ways to do this. The actual steps can vary (sometimes greatly) depending on which version of Windows Server is running.

Delving into those steps or describing the drawbacks and advantages of each is beyond the scope of this article. Here, I'll simply point out that to properly use logging to debug your applications, you must turn it on before the actual errors occur. You'll find a lot of useful information in these two MSDN articles for Windows Server 2003 and 2012: "How to configure Web site logging in Windows Server 2003" (bit.ly/cbS3xZ) and "Configure Logging in IIS" (bit.ly/18wSgt). If these don't meet your needs, there are plenty

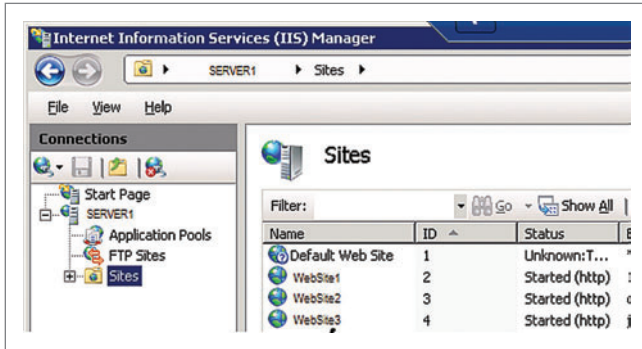


Figure 1 Finding the Web Site ID Number

of other online articles relevant to turning on logging in IIS for other versions of Windows Server.

Determine the Proper ID Number

Once logging is on, you need to find out the ID number in IIS of the Web site you're troubleshooting. This is crucial, as servers typically host more than one Web site, and trying to find the log folder manually can be daunting. (I attempted it on a server running 45 Web sites and it was almost impossible.)

Open IIS Manager to display all Web sites being hosted. For this example, suppose I'm attempting to find out why WebSite2 suddenly stopped working or is only working intermittently.

As you can see in **Figure 1**, the ID for WebSite2 is "3." The next step is to open the corresponding log folder, which is typically (but not always) found in the Inetpub folder. Windows usually creates this folder in the root (C:) of the server, but in my case, the Inetpub folder is on the D: drive. Industry best practices advise keeping code and OS drives separate for easy swapping in case of failure.

IIS usually keeps many different files, depending on how you've configured the server history or how long logging has been on.

Windows names all logging folders W3SVC#, where # is the ID of the given Web site. Because the ID of the site being debugged in this case is 3, the log files are located in a folder named W3SVC3, as shown in **Figure 2**.

Browse the Files

When you open the correct folder, you may see a lot of files. IIS usually keeps many different files, depending on how you've configured the server history or how long logging has been on. To find the file you need, an almost-sure bet is to scroll to the bottom of the list and open the last file, though if you have the exact time the error occurred, you can locate the file using the date and time

in the name of the file. Either way, open the file using a file editor such as Notepad.exe.

You'll probably see a lot of data in the file. At first glance, the information might seem cryptic and useless, but with a little study you can find plenty of gems buried within this data. I'll look at some of the most useful data elements recorded by the logging process.

IIS and Windows log an individual line for each HTTP request. A typical line looks like this:

```
2013-08-28 00:01:12 128.128.128.20 GET /default.asp - 443 -
200.200.200.17 Mozilla/4.0+(compatible; +MSIE+8.0; +Windows+NT+6.1;
+WOW64;+Trident/4.0;+SLCC2;+.NET+CLR+2.0.50727; +.NET+CLR+3.5.30729;+.
NET+CLR+3.0.30729;+InfoPath.3;+MS-RTC+LM+8; +.NET4.0C;+.NET4.0E;
+.NET+CLR+1.1.4322) 200 0 0 15
```

At first glance, the information might seem cryptic and useless, but with a little study you can find plenty of gems buried within this data.

This is a line from an actual IIS log. The data shown here is in one of the "standard" formats. However, because this option is highly configurable, there's no guarantee your files will look exactly like my sample. Instead of wading through all the data, I'm going to focus here on the elements that are of most interest to debugging an application.

The first bolded element in the sample is the date of the request. Keep in mind this is the server date. As many Web applications run worldwide across many servers deployed in different time zones, this date can be misleading. Make sure the date accurately reflects the actual time the errors occurred. Many servers use GMT time, but you should validate the format.

Next, you'll see the IP address that was accessed, the HTTP operation type (GET) and the actual file requested or accessed. In the following sample line, the code is calling the default.asp file:

```
128.128.128.20 GET /default.asp
```

This information is valuable, as there may already be errors occurring at this stage of the process. For example, you might be expecting another file to be executed at this point.

The next part of the line shows the IP address where the request originated, as well as the receiving port:

```
443 - 200.200.200.17
```

This information is also important, as sometimes it's necessary to verify that the request you're troubleshooting actually came from a known source.

As you can see, the actual port used is also referenced. This seemingly unimportant bit of information is vital when looking for issues. For example, the firewall may not be configured correctly. This data is followed by a lot of information, mostly related to versions:

```
+MSIE+8.0; +Windows+NT+6.1;+WOW64;+Trident/4.0;+SLCC2;
+.NET+CLR+2.0.50727; +.NET+CLR+3.5.30729;+.NET+CLR+3.0.30729;
+InfoPath.3;+MS-RTC+LM+8; +.NET4.0C
```

For example, you can see whether the browser is running 32- or 64-bit, the CLR versions (for those who delve that deeply into the .NET universe), and the actual .NET version installed on the server (in this case, .NET 4 C).

Get to the Bottom of Things

Up to this point, I've shown the relatively obvious parts of the log file entry. Most importantly, you can see which browser is responding to the HTTP request. Sometimes this will suffice, as different browsers may produce different results. Here are some partial strings showing what Firefox and Chrome might look like in the file:

```
;+rv:17.0)+Gecko/20100101+Firefox/17.0 404 0 2 109
```

```
+AppleWebKit/537.36+(KHTML, like+Gecko)+Chrome/28.0.1500.95+Safari/537.36  
200 0 0 125
```

It can be difficult to detect which of several HTTP requests is being debugged because they all look similar. This is where changing the browser can come in handy. By adding an entry from a different (and unexpected) browser, such as Safari or Opera, it can be easier to find and consequently troubleshoot the entry in question.

Finally, take a look at the last four items on the line:

```
200 0 0 15
```

The last number, 15, is the response time (in milliseconds) of the HTTP request. This is a very useful piece of information. Knowing how long a request took to process can help you decide whether a given code snippet, Web service or process is taking the desired or “normal” amount of time.

It may surprise you to discover
relatively simple steps in a
process are taking unexpected
processing time.

It may surprise you to discover relatively simple steps in a process are taking unexpected processing time. In a recent case, an application sometimes logged in and sometimes failed to log in, without producing a trappable browser error—or any error at all. It just failed. After reviewing the response times in the log, the developers discovered this property in the web.config file:

```
<add name="CacheProfile1" duration="30" />
```

The value of this parameter, apparently harmlessly set to 30 seconds, was simply too large. Once it was reduced, the application worked as expected.

Now (repeating the line here for clarity) I'll zero in on one of the most important parameters from the set I'm reviewing. The first item, 200, is the actual HTTP response from IIS:

```
200 0 0 15
```

That HTTP response code, 200, means success. Often, you'll encounter a known error type, such as 404 (not found) or 500

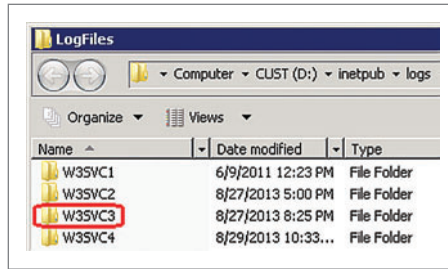


Figure 2 Opening the Log-File Folder

(internal server error) and that may give you enough information to troubleshoot and resolve an issue. For an official list of HTTP status codes, visit bit.ly/17sGpwE.

I'll now relate one more real-world case—the one that actually prompted me to share this article. I had a Web site that worked and executed perfectly on PCs, but as soon as the users accessed the site on their iPad devices, video streaming simply didn't work. To make things worse, there

was no error code; the video just refused to function.

This is where troubleshooting with the logs proved invaluable. By checking the logs and verifying the HTTP request was being made from Safari (to isolate the request), I discovered the server reported a 404 error. The error message was confusing and the error itself seemed implausible, as the PC version of the site worked perfectly.

Though the logs were reporting the object wasn't being found, I knew very well the files were in place. This prompted me to review the different ways iOS and Windows handle and store files. As I explored the source code that loaded the video, I discovered that the actual path to the video files had been hard-coded in the source, and that the path didn't exist in iOS/iPad devices. That was the reason for the 404.

It's important to note here that all the symptoms pointed elsewhere. For example, such a problem is typically resolved by checking for the existence of unsupported media types (or Multipurpose Internet Mail Extensions [MIMEs]) in IIS. However, if the problem were a missing MIME type, the error code would have been an HTTP 415 (unsupported Media type) or a similar error, and the logs didn't report that. Debugging using the IIS logs proved instrumental in finding the problem. I saved a significant amount of time by seeing the actual error code and investigating it, rather than chasing after what would have initially seemed more likely. Once again, knowing how to read the logs saved the day.

Wrapping Up

Log files can be a powerful tool for debugging and troubleshooting applications, even in “blind” situations, provided you know where to find them and what the data means. Investigating the data in the logs is one of the simplest—yet more sophisticated and complete—methods you'll find to resolve issues.

It takes a little practice and certainly some discipline, but once you're comfortable with these techniques, you should be able to debug and resolve most application and IIS issues. I've put these techniques to good use, and so can you. ■

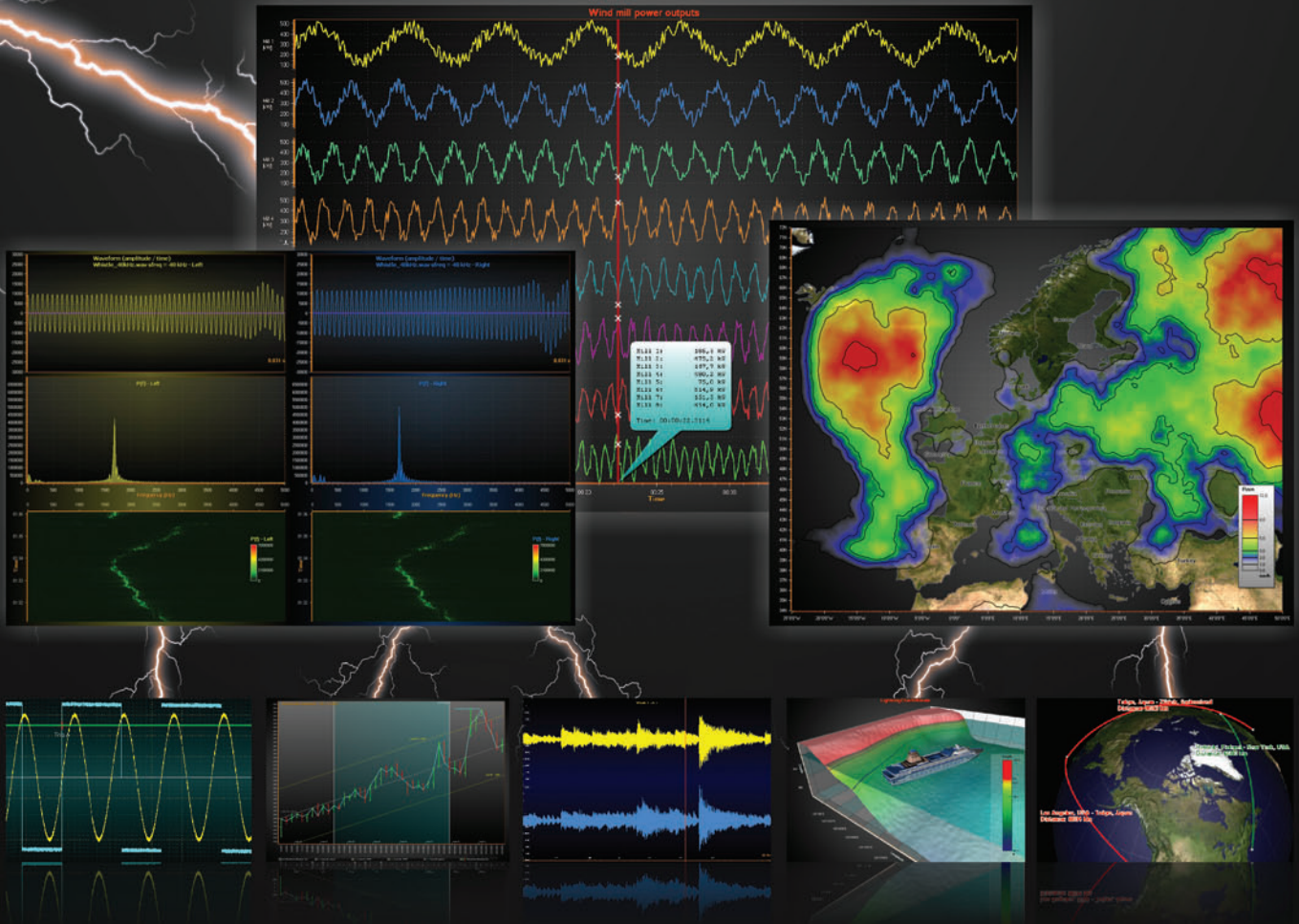
EDUARDO SANABRIA is a service delivery consultant IV for HP Enterprise Services in El Paso, Texas. There he delivers quality results that matter, serving as the .NET subject matter expert (SME) in his current project. Sanabria brings to Hewlett-Packard Co. more than 25 years of full-cycle application development experience. His specialties are .NET, database applications, data processing and Web development. Sanabria can be reached at EdSanabria@Yahoo.com.

THANKS to the following technical expert for reviewing this article:
Roger Hawkins (Hewlett-Packard)

ULTIMATE CHARTING POWER

LightningChart

The fastest rendering data visualization components
for WPF and WinForms



- Fully DirectX accelerated
- Superior 2D and 3D rendering performance
- Very extensive property sets
- Optimized for real-time data monitoring
- Supports gigantic data sets
- Professional, friendly and fast customer support
- Compatible with Visual Studio 2005...2012

WPF charts performance comparison

Opening large dataset	LightningChart is up to 977,000 % faster
Real-time monitoring	LightningChart is up to 2,700,000 % faster

Winforms charts performance comparison

Opening large dataset	LightningChart is up to 37,000 % faster
Real-time monitoring	LightningChart is up to 2,300,000 % faster

Results compared to average of other chart controls. See details at www.LightningChart.com/benchmark. LightningChart results apply for Ultimate edition.

**FREE
TRIAL**

Download a free 30-day evaluation from
www.LightningChart.com

Arction
Pioneers of high-performance data visualization

Unit Testing SQL Server OLAP Cubes Using C#

Mark Nadelson

I feel a little like Thomas Jefferson when I say, “We hold these truths to be self-evident, that all code has certain unalienable rights and that among these rights is the ability to be thoroughly unit tested in a simple and concise way so bugs can be easily identified and production outages minimized.” A bit dramatic, yes, but it gets my point across.

Of course, most developers believe their code should be unit tested, but what happens when the definition of “code” is blurred? This was the situation I found myself in recently when presented with a complicated issue and tasked to find a solution. A group of developers was involved with writing a complex online analytical processing (OLAP) cube using SQL Server Analysis Services (SSAS). This cube had numerous dimensions all tied to an extremely complex

fact table. Because the developers were quite skilled at developing cubes, they were able to piece the cube together, but validating the results of their Multidimensional Expressions (MDX) queries was a daunting task. The difficulty was compounded by a number of factors, including the amount of data in the dimensions and fact table, as well as the time and computing resources needed to build the cube. Once the cube was built, the results (produced by MDX queries) were sent to the users. If the users found an issue with the data, it would take a long time to track the problem down. Also, once the underlying problem was discovered and fixed, the cube would need to be regenerated. To make matters worse, if dimensions were added, the underlying fact table was updated or the cube was built using different aggregations, there was no way to determine the full effects of these changes. A seemingly innocent change could have a far-reaching and cascading effect on queries to the cube.

The solution for the cube conundrum is to create an environment and a process wherein you can stage the cube’s dimensions and facts using a limited amount of data, and execute queries against the cube using the production version of the cube’s schema. Ideally the test version of the cube would be created from scratch each time the unit tests run, eliminating the chance for side effects occurring from pre-existing artifacts. Other requirements for the unit-test framework (and really these apply to most unit-test frameworks regardless of the target application) are to ensure the test validations are repeatable, and if failure occurs to quickly and easily pinpoint why the test failed (having the test case say “failed because data did not match” is less than ideal).

This article discusses:

- Cube characteristics
- An example cube to analyze sales
- Creating a unit-test version of the cube
- Validating the cube
- Testing the cube

Technologies discussed:

SQL Server, C#, Online Analytical Processing (OLAP)

Code download available at:

msdn.microsoft.com/magazine/msdnmag0114

In this article, I'll present a framework that lets you create a suite of unit tests to validate the MDX-based output of an OLAP cube. The architecture described enables the creation of a cube using an existing schema, within its own unit-test database, recreated each time the test suite is run. It also lets you provide MDX queries that are executed against the newly formed unit-test version of the cube. Further, it validates the results against a pre-existing template and presents them in simple HTML format. This pattern of test case validation against a template can be extended to unit-testing frameworks in which the data results are large and complex.

Cube Overview

Before delving into the solution for the cube-testing issue, I'll briefly go over the concepts and components that comprise a cube. Cubes are a means of quickly accessing data held within a data warehouse. Cubes organize and summarize the data into a multidimensional structure. Cubes are the main component of OLAP technology, and they provide an easy-to-use mechanism for querying data with quick and predictable response times. A cube is made up of the dimension data and measures (numeric facts). The central table in a cube is known as the fact table, and it's the source of the cube's measures. The dimension tables are referenced by the fact table, and they contain hierarchical levels of information that can be queried. The dimension hierarchy lets users ask questions at a high level. Then, using the dimension's hierarchy, users can get to more details.

Cubes are contained within a database. The objects that make up the cube structure for a given database are as follows:

- **Data sources:** These are the sources of the information to be loaded into the cube.
- **Measures:** These are the numeric values represented in the cube. They can be dates, but are usually numeric with differing levels of aggregation (such as Sum, Max, Min and Count).
- **Dimensions:** These are attributes associated with measures. Business data, customer names and geographic regions are common examples of dimensions.
- **Partitions:** A partition defines a portion of the fact data loaded into a measure group. By creating multiple

partitions, the cube can be processed in parallel and stored and queried separately, thereby improving performance. You can also reprocess individual partitions without affecting other partitions.

- **Cube roles:** Each cube should have at least one cube role to allow access to end users. Roles can allow access to all data or a subset of the data stored within the cube based on an individual user ID or an Active Directory group.

Cubes are the main component of OLAP technology, and they provide an easy-to-use mechanism for querying data with quick and predictable response times.

The definition for a cube's schema can be extracted from SSAS in the form of XML for Analysis (XMLA). XMLA is a SOAP-based XML protocol that gives access to the cube over HTTP. The XMLA definition contains all the details for each of the five cube objects described earlier. XMLA lets you recreate the cube on different databases or servers quickly and easily. It's the cornerstone by which the unit-testable cube is created.

Once a cube is created and processed, the data measures can be queried using a mix and match of the variety of dimensions used to create it. Cubes are queried with the aforementioned MDX syntax. Like a SQL query, an MDX query contains a data request (using the SELECT statement), a data point (using the FROM statement) and an optional data filter (using the WHERE clause). Here's a basic example:

```
SELECT ([[axis specification using Measures>]]...) ON AXIS(0),
      ([[axis specification using a dimension hierarchy>]]...) ON AXIS(1)
FROM [cube>]
WHERE (<filter specification>)
```

The Example Sales Cube

I created an example cube that lets you quickly query sales data for various stores on various dates by various customers (see Figure 1).

The cube is made up of four dimension tables connecting to a fact table. For simplicity, the fact table contains one measure called Quantity, which represents the amount of a given item sold to a given customer at a store location on a given purchase date. With this cube configuration, the user can quickly query for various facts using various dimensions. For example, corporate executives can get a clear idea of which products are selling at which stores, or they can delve into details such as which items sell best during a particular year or month. Having a large number of dimensions lets an executive view sales data in a variety of ways, providing better insight into the performance of stores.

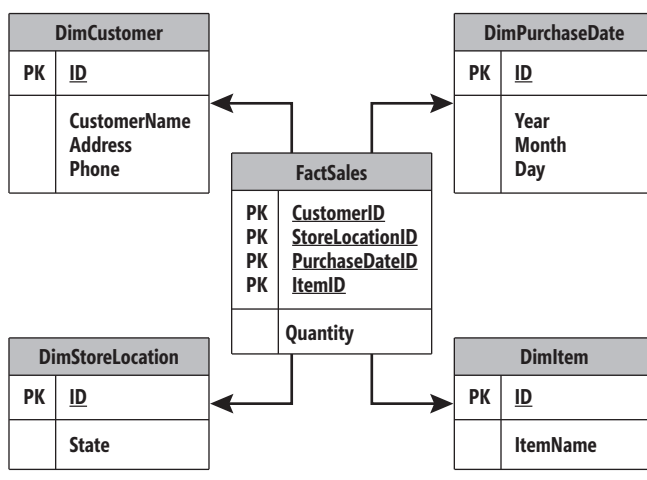


Figure 1 Example Sales Cube Database Diagram

Figure 2 The Layout of the HTML Representation of an MDX Query Result

		Column Dimension Caption 1 (Value 1)	Column Dimension Caption 1 (Value 1)	Column Dimension Caption 1 (Value 2)	Column Dimension Caption 1 (Value 2)
		Column Dimension Caption 2 (Value 1)	Column Dimension Caption 2 (Value 2)	Column Dimension Caption 2 (Value 1)	Column Dimension Caption 2 (Value 2)
Row Dimension Caption 1 (Value 1)	Row Dimension Caption 2 (Value 1)	MDX Result Value	MDX Result Value	MDX Result Value	MDX Result Value
Row Dimension Caption 1 (Value 1)	Row Dimension Caption 2 (Value 2)	MDX Result Value	MDX Result Value	MDX Result Value	MDX Result Value
Row Dimension Caption 1 (Value 2)	Row Dimension Caption 2 (Value 1)	MDX Result Value	MDX Result Value	MDX Result Value	MDX Result Value
Row Dimension Caption 1 (Value 2)	Row Dimension Caption 2 (Value 2)	MDX Result Value	MDX Result Value	MDX Result Value	MDX Result Value

Microsoft.AnalysisServices.Server. A call to `Server.Connect(<Connection String>)` using the unit-test server name passed in establishes the connection (see **Listing 2** in the code download). Once the server connection is successfully established, the unit-test version of the cube database is removed if it already exists. This removal is done with the `DropDatabase` method, which is passed the connected `Server` instance and the name of the unit-test database to drop (`TargetServerName`). `DropDatabase` ensures that the server instance is connected, looks up the `Microsoft.AnalysisServices.Database` using the unit-test database name passed in and, if the database exists (the database instance is not null), the database is dropped (see **Listing 3** in the code download).

Creating the Unit-Test Version of the Cube

As mentioned, the cube definition can be extracted from SSAS in the form of an XMLA document. This XMLA document is used to create the unit-test version of the cube. Using the production definition of the cube ensures the tests will accurately exercise all features of the cube in question. You interface with the SSAS engine using `Microsoft.AnalysisServices.dll`, which can be found in the SQL Server SDK assemblies.

The object used to generate the cube is `XMLAtoCube`. The constructor takes in a base configuration directory under which the XMLA is stored. The directory structure I chose to house the cube XMLA is `<base directory>\<production server name>\<production database name>`.

`XMLAtoCube` contains one public method, `CreateCubeFromXMLA`, which takes the following parameters (see the method's code in **Listing 1** in the Listings.zip file in the accompanying code download):

- **SourceServerName:** The name of the production server that contains the cube.
- **TargetServerName:** The name of the server that will house the unit-test version of the cube.
- **SourceDatabaseName:** The name of the production database that contains the cube.
- **TargetDatabaseName:** The name of the database that will house the unit-test version of the cube.
- **DataSourceProviderDefinition:** The connection string URL that points the unit-test version of the cube to the location of its source dimensions and fact table. This will be the source of the scaled-down data for the unit test.

`CreateCubeFromXMLA` first establishes a connection to the version of the unit-test analytics server and drops the unit-test version of the cube database if it already exists. The dropping of the database is important because it ensures the tests are conducted on a clean environment without any residual artifacts contaminating the outcome. The connection to the analytics server is executed within the `ConnectToServer` method using an instance of

The next step is to take the original cube's XMLA definition and generate the unit-test version. The unit-test version contains the same dimensions, dimension hierarchies, measures, partitions, roles and so on as the original cube. The difference is that it's generated in a new database pointing to a different location for its source data. The `AddCubeToDatabase` method creates the test cube (see **Listing 4** in the code download). `AddCubeToDatabase` reads the XMLA definition from the file system using the naming convention mentioned earlier. The XMLA file name is passed to an instance of `XmlTextReader` at construction. The XMLA is read in using the `Microsoft.AnalysisServices.Utils.Deserialize` method, which is passed the `XmlTextReader` and a newly created instance of `Microsoft.AnalysisServices.Database`. The `Database` object instance now contains the complete definition of the cube, but that definition still has the original cube's database name and data source. Pointing to the unit-test database simply involves setting the `Name` and `ID` properties of the "Database to unit-test database name" (`targetDatabaseName`) parameter. This database can then be added to the instance of the unit-test analytics server by calling `Server.Databases.Add(<unit test database>)` followed by the `Database.Update` method.

After the database has been created, you need to update the cube's data source to the external unit-test data collection. The `Database` instance has a list of `DataSource` instances (usually just one data source is associated with a cube) and the unit-test connection string is used to replace the connection string contained within the XMLA definition. After the connection string is replaced, a call to the `DataSource.Update` method updates it within the SSAS server. At this point, the customization of the XMLA definition is completed and the remaining pieces of the cube (`DataSourceView`, `Dimension` and `Cube`) are updated and processed.

After the call to `AddCubeToDatabase` is complete, the unit-test version of the cube has been created within the specified server using the chosen unit-test database. It points to a custom set of source data. Although the cube has been created, it's still empty. In order

to populate the dimensions with the source data, the dimensions must be processed. The `ProcessDimensions` method is called and passed the Database instance. All the Dimensions within the Database are processed using the `Dimension.Process(ProcessType.ProcessFull)` method. Once the dimensions have been successfully processed, the cubes within the unit-test database are processed using the `ProcessCube` method. Like `ProcessDimensions`, `ProcessCube` takes the Database instance, loops through all the cubes in the Database and calls `Cube.Process(ProcessType.ProcessFull)` (see **Listing 5** in the code download). At this point, the unit-test cube has been created and populated with the targeted test data. The next step is to run tests against it to ensure that the cube behaves as expected.

Validating the Cube

Although the solution for validation is targeted for the cube, the design pattern being used can apply to other unit-testing frameworks as well. The pattern employed is testing using template validation. Simply put: When the test runs, a data structure containing the results is created and validated against a previously stored version of the data structure that was deemed correct. Because it's hard to validate a binary data structure, an HTML representation of the structure is presented to the unit tester. The tester uses the HTML representation to validate the initial test results of the unit test (to make sure the results match what is expected) and to examine what caused a unit test to fail during subsequent runs. In a failure, the HTML displays the original data structure as well as which piece of the data structure failed validation and why. This is critical to helping debug the issue.

The best way to test most scenarios is using the “black-box” testing methodology. A black-box test passes input and validates output. Because the output for a cube is a query result, the input is the query. You use MDX statements to query a cube. After the cube interprets the MDX query, it returns a result. The result is in the form of rows and columns that are a representation of the dimensions chosen for running the MDX query. The MDX query result can be quite complex because the query can involve many

dimensions, resulting in a variety of rows and columns in the output. The data structure chosen to hold the cube data is a Dictionary of `CubeRow` instances keyed by the name of the cube row. The `CubeRow` class contains two alternative data structures—the one used depends on the situation. One data structure is a Dictionary of `CubeTuple` instances keyed by the name of the cube column. The `CubeTuple` is simply an object that contains the cube's column name and the given column's value. The Dictionary of `CubeTuple` objects is used when the given cube column contains a value. The second data structure is another Dictionary, mapping a row name to a `CubeRow` instance. Because an MDX query can have many levels of row dimensions, `CubeRow` contains its own Dictionary of row name and `CubeRow` instances.

Not only are the results of the cube stored in a `Dictionary<String, CubeRow>` instance, the results are also stored in an HTML string. The HTML string allows the tester to have a visual representation of the cube results. The HTML table contains multiple levels of column and row headers, and the HTML table cells contain the MDX values. **Figure 2** shows the layout of the HTML representation of an MDX query result.

`BaseMDXTest` contains the code for executing an MDX query and building the MDX result data structure as well as the representative XML. `BaseMDXTest` uses `Microsoft.AnalysisServices.AdomdClient.dll`, found in the SQL Server SDK assemblies, to connect to the SSAS cube and execute the MDX queries. `BuildTemplate` is the method that runs the MDX query and builds the MDX result Dictionary as well as the HTML representation. First, a connection to the cube is established. In order to establish and open a connection, the connection string is passed to an instance of `MicrosoftAnalysisServices.AdomdClient.AdomdConnection`. The `Open` method is then called on the newly created connection instance and the connection instance is returned to the caller. Once a connection is created, an instance of the `MicrosoftAnalysisServices.AdomdClient.AdomdCommand` method is established and is passed the MDX query string and the `AdomdConnection` instance. A call to the `AdomdCommand.ExecuteCellSet` method executes the MDX query against the unit-test version of the cube and returns a `MicrosoftAnalysisServices.AdomdClient.CellSet` instance (see **Listing 6** in the code download).

After the `CellSet` instance is retrieved, a check is made to ensure that the result set has two axes. Axis 0 contains the columns and Axis 1 contains the rows. Each axis contains a collection of `Position` objects. A `Position` represents a tuple on the given axis and contains one or more `Member` objects. A `Member` represents the column or row headers for the given `Position` (see **Listing 7** in the code download).

Next, the number of rows and columns returned by the MDX query is computed. This is done by taking the number of rows (the count of `Position` objects on Axis 1) plus the number of column dimensions (`CellSet.Axes[0].Positions[0].Members.Count`). The number of columns is added to the rows because when representing the MDX results as a two-dimensional table, the columns are included within the set of rows. Likewise, the number of columns is computed by taking the number of `Position` objects on Axis 0 plus the number of row dimensions (see **Listing 8** in the code download).

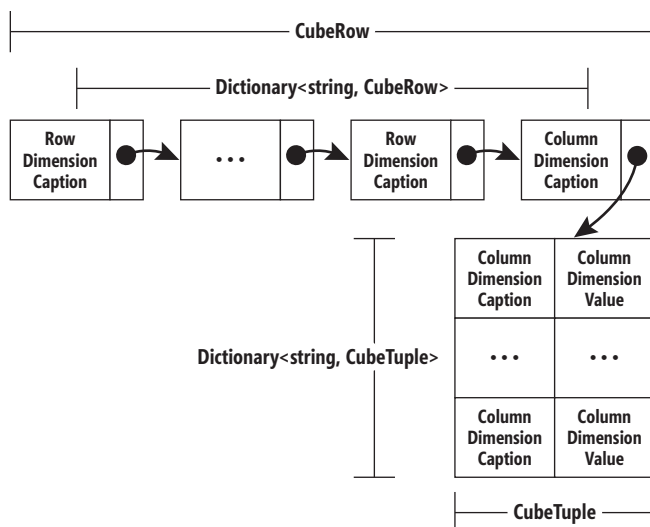


Figure 3 A Graphical Representation of the CubeRow Dictionary

Given the number of rows and columns, the Dictionary of CubeRow objects can be generated as well as the HTML representation of the MDX output. **Listing 9** in the code download contains the code for traversing the MDX result set, creating the HTML and storing the results in the CubeRow Dictionary. First, the number of rows is looped through. If the row number is greater than the number of column dimensions, then it's known that a new row of MDX results is available. In other words, when the rows of column headers have been passed, the MDX data is available. At this point, a new CubeRow object is created.

The next step is to loop through each column within the row. If the current row number is lower than the number of column dimensions, then the current row is actually a row of column headers. For the Dictionary, the header captions are concatenated for each column location, separated by a colon. This means that if a header is made up of multiple dimensions, each column will be concatenated with the given dimension in descending order of resolution. The HTML generation for a column header is more straightforward. The dimension caption is simply surrounded by the HTML table header tags (<th></th>). For each case, the current dimension caption is retrieved by getting the header axis (CellSet.Axis[0]) and accessing the column position (current column count minus the current row dimension count) and the current Member within that Position (current row count).

Test Title

TestPurchaseCubeMultipleRowsAndColumns

MDX Query

```
select {{EXCEPT([PurchaseDate].[Year].members, [PurchaseDate].[Year].[All])}*
{EXCEPT([PurchaseDate].[Month].members, [PurchaseDate].[Month].[All])}*
{EXCEPT([PurchaseDate].[Day].members, [PurchaseDate].[Day].[All])}} on 0,
{{EXCEPT([StoreLocation].[State].members, [StoreLocation].[State].[All])}*
{EXCEPT([Item].[ItemName].members, [Item].[ItemName].[All])}} on 1 from [PurchaseCube]
```

Original Template

		2013	2013
		8	8
		1	2
CT	Food	50	
CT	Shoes	100	
NY	Food		
NY	Shoes		50

Test Results

		2013	2013
		8	8
		1	2
CT	Food	50	
CT	Shoes	100	
NY	Food		
NY	Shoes		50

Test Passes!

If the current row number is greater than the number of column dimensions, then the column headers are no longer being processed. Instead, the MDX result set row tuples are next in line for processing. Similar to columns, which have headers, MDX result set rows may also have headers. If the column number being processed is less than the number of row dimensions, then a row header is being processed. For each row header, a new Dictionary<string, CubeRow> is created, added to the current Dictionary and set as the current MDX result Dictionary. What this means is that for each row header, there exists a Dictionary of rows that contains more granulated MDX result data.

Extracting the row header caption is similar to extracting the column header caption. The row caption is retrieved from the Member object at the current column location from the Position at the current row from CellSet.Axis[1]. The row caption is the key to the Dictionary of CubeRows, and if the current CubeRow Dictionary doesn't have the extracted row caption, the CubeRow object is added to the Dictionary keyed by the row caption. If the row caption does exist within the current CubeRow Dictionary, the CubeRow object is retrieved and set as currentCubeRow.

After the row caption members have been exhausted (that is, the current column count is greater than the number of row dimensions) and the column header rows have been traversed (that is, the current row count is greater than the number of column dimensions), it's time to add MDX cell values to the current CubeRow object. Each combination of a column header and column value is considered to make up a single CubeTuple instance. Each CubeRow contains a Dictionary of CubeTuple objects keyed by the column header. The column header is retrieved from an array of column headers previously constructed (recall a column header is a colon-delimited string of all column captions concatenated together). The index of the column header is the current column count minus the number of row dimensions (the total column count includes the row dimensions). The current MDX CellSet value is retrieved by accessing the appropriate two-dimensional (column, row) point. This is based on the current column count (minus the number of row dimensions) and the current row count (minus the number of column dimensions). This value is added to the CubeRow object using the AddTuple method, passing it the column header and column value. At the same time, the HTML representation is updated by adding the MDX cell value in between HTML table dimension (<td></td>) tokens. See **Figure 3** for a graphical representation of the Dictionary<string, CubeRow>.

Both the Dictionary and HTML representation of the template are persisted to a previously defined file location using the BaseMDXTest.PersistTemplate method. Because the template needs to be manually validated by the unit-test developer, the test is considered to have failed, which is why the BaseMDXTest.TestMDXQuery method returns false for success.

Once the template has been created, subsequent runs of the same test are validated against the previously stored template. When the TestMDXQuery method is called, a check is first made to see if a test with the given name exists. If it does and a new template creation isn't requested (requesting to recreate the template may occur if the current template is incorrect), then the test result template is

Figure 4 The HTML Output of an Example Test

loaded into memory. The template includes both the object representation and HTML representation of the MDX result set. The `BaseMDXTest.RunComparison` method executes the MDX query and compares the results against the stored template. The MDX query results are traversed in the same way as they were during the template creation. The main difference between the creation of the original template and the validation against the template is that instead of creating the `Dictionary<string, CubeRow>`, lookups are done against the template `Dictionary`, checking to see if the same MDX query results exist. When looping through the rows and columns, the HTML table is created the same way as during template creation, except now the cells within the HTML table are colored. A green cell indicates that the cell matches the original template; a red cell shows a mismatch. By coloring the cells and presenting them in an HTML table, the unit tester has an immediate view of why the test case passed or failed. Anytime a mismatch is found, a Boolean (`testPass`) is set to false to indicate the test case failed.

Although it isn't
straightforward, even an OLAP
cube can be unit tested.

While traversing the MDX query results and validating them against the template `Dictionary`, each `CubeTuple` (an object that contains the column's dimension, concatenated names and the column's value) found is removed from the current `CubeRow` `Dictionary` of `CubeTuple` objects. Therefore, after the entire MDX query result is passed, the original template `Dictionary` should have `CubeRow` objects with an empty `Dictionary` of `CubeTuple` objects if the MDX result was a complete match. Otherwise the new MDX query result had missing data that was contained within the original result. The `BaseMDXTest.CheckForExtraDataInTemplate` method examines the template `Dictionary` for remaining `CubeTuple` objects, executing recursively, and returns a value of true if `CubeTuple` objects remain. The `testPass` Boolean within the `RunComparison` method is set to false if extra data is found, and the test case fails.

After the MDX results have been completely traversed and validated against the template `Dictionary`, an instance of the `CubeComparisonResult` object is returned. It's constructed with the `testPass` Boolean and the HTML table showing the result. The `BaseMDXTest.TestMDXQuery` method uses `CubeComparisonResult` to build an HTML page showing the original MDX query result HTML table and the comparison HTML table. The HTML is persisted to the file system by executing a call to the `BaseMDXTest.PersistTestReport` method, which creates a `TestReport.html` summary Web page listing all test runs and links to their HTML result pages, as well as a summary of the number of test cases that passed and failed.

Testing the Purchase Cube

Using both components of the cube-testing framework—the cube-creation code (`XMLAtoCube`) and the MDX query result template (`BaseMDXTest`)—you can create unit-test cases that

validate the cube. Although the code for the framework is extensive, creating the test cases is simple and straightforward. **Listing 10** in the code download contains sample unit tests for validation of the Purchase cube. These test cases use the Microsoft testing framework, but any testing framework can be incorporated.

The unit-test object (`PurchaseCubeTest` in the example) inherits from `BaseMDXTest`. The default constructor of `PurchaseCubeTest` constructs `BaseMDXTest` with the URL of the SSAS server where the cube is located and the base directory in which to store the MDX query result template and subsequent test results.

A `[TestInitialize]` method is used to create the unit-test version of the Purchase cube. It uses the XMLA definition of the original cube and creates it on the unit-test SSAS server (`targetServerName`) using a unit-test database (`targetDatabaseName`). It also points the source data URL to the location of the test dimension and fact data. `[TestInitialize]` is run only once for a given `[TestClass]`, which ensures the cube is created only at the start of testing.

The test cases themselves are executed within `[TestMethod]` annotated methods. Each test case is simple. The MDX query is defined and then executed using the inherited `BaseMDXTest.TestMDXQuery` method, naming the test case and passing it the MDX query. `TestMDXQuery` returns true if the test passes or false if it doesn't, and an `Assert.IsTrue` method is used to pass or fail the unit test. After all tests have been run, the resulting HTML test document can be opened and the failing test cases can be examined. **Figure 4** contains an example HTML output of one of the tests.

Properly Tested Code

Although it isn't straightforward, even an OLAP cube can be unit tested using C#. The inclusion of both the `Microsoft.AnalysisServices.dll` and the `Microsoft.AnalysisServicesAdomdClient.dll` files within the SQL Server assemblies provides you with the APIs for both creating and querying SSAS cubes. The architecture presented lets you add a rich set of unit tests that produces output in a readable format so test case failures can be quickly identified. The method of test-case validation by template can be applied to other unit-test architectures where output validation isn't straightforward. Examples of these range from applications that rely on traditional relational database persistence where the template contains the data expected to be stored in the database after the application runs, to UI applications that store the state of the UI in a data structure and display the UI in an HTML form.

I'm not quite sure if Thomas Jefferson or our founding fathers would compare the freedoms of a nation to the rights of properly tested code, but I'm pretty sure your users and supervisors would be pleased to know your application has been correctly put through its paces. ■

MARK NADELSON is a professional software developer with 22 years of experience in the telecommunications, Internet and finance industries. Throughout his career he has used a number of programming languages including assembly, C, C++, Java and C#. Nadelson is also the author of two books and a number of technical articles.

THANKS to the following technical experts for reviewing this article:
David Neigler and Joe Sampino



Frequent Item-Sets for Association Rule Learning

An important task in the areas of machine learning and natural user interfaces (NUIs) is extracting frequent item-sets from a list of transactions. The problem is best explained by example. Suppose you have a list of items purchased by different customers in a supermarket. For instance, one customer's transaction might be (apples, bread, celery) and another's might be (bread, donuts, eggs, lettuce). There could be many thousands of these transactions. An item-set is a subset of all possible items, such as (apples, bread). The problem is to extract from the list of all transactions just those item-sets that meet some minimum count. Suppose the minimum count is 20 and item-set (apples, bread) occurs in the list of transactions 25 times; therefore, (apples, bread) would be a frequent item-set.

Identifying frequent item-sets can be useful in many ways. In the case of a supermarket, knowledge of which items are often purchased together could be used for product placement, targeted marketing, and so on. Additionally, once all frequent items sets have been identified, these item-sets could be analyzed further to extract rules such as, "if customers purchase both apples and bread, then there's high probability they will also purchase lettuce and milk." This overall process of first extracting frequent item-sets and then harvesting if-then rules is called association rule learning.

This article explains in detail the frequent item-set extraction process. The problem is surprisingly difficult and has been the subject of quite a bit of research. The best way to see where this article is headed is to take a look at the demo program in **Figure 1**.

The demo program sets up a dummy list of 10 transactions. Behind the scenes, there are a total of 12 items in a hypothetical supermarket: apples, bread, celery, donuts, eggs, flour, grapes, honey, icing, jelly, kale and lettuce. The first transaction is (apples, bread, celery, flour). The demo then shows how raw transactions can be encoded into 0-based integers for efficient processing. Because apples maps to 0, bread to 1, celery to 2, and so on, the first transaction is encoded as (0, 1, 2, 5).

The algorithm to extract frequent item-sets requires three input parameters. The first parameter is a minimum threshold value necessary for an item-set to be classified as frequent. This is usually called the support value. In the demo, the support value is set to 0.30, which means that $0.30 * 10 = 3$ is the minimum number of times an item-set must occur in the list of all transactions for the

item-set to be added to the result list of frequent item-sets. Different problem domains will have different meaningful support values.

The second input parameter is the minimum size of an item-set. In the demo that value is set to 2, so an item-set of (apples, donuts) is examined but an item-set of just (apples) is not. The third input parameter is the maximum size of an item set. Here that value is set to 4, so only item-sets with up to four items, such as (apples, celery, eggs, grapes), are examined while larger item-sets such as (apples, bread, donuts, flour, honey) are not.

The demo program calls the method to extract frequent item-sets from the list of transactions and finds a total of 12 frequent item-sets. The first frequent item-set is (0, 1), which occurs five times in the transaction list. The last frequent item-set is (0, 1, 2, 5), which occurs three times. The demo concludes by displaying the frequent item-sets in string form.

This article assumes you have at least intermediate-level programming skills, but does not assume you know anything about association rule learning. The complete demo program code is presented in this article, and it's also available as a download from msdn.microsoft.com/magazine/msdnmag0114. The demo is in C# but you shouldn't have too much trouble refactoring to another language, such as Python. All normal error-checking has been removed to keep the size of the code smaller.

Why Is Frequent Item-Set Extraction a Difficult Problem?

At first thought, extracting frequent item-sets from a list of transactions doesn't seem so difficult. A simple brute-force approach would be to simply generate all possible item-sets and iterate through the transactions for each candidate, counting the number of times the candidate appears to see if that item-set meets the minimum support count. Unfortunately, in all but artificially small problems, the number of possible item-sets is unimaginably large. For example, a typical supermarket might have far more than 10,000 distinct items in stock. The number of item-sets of size 2 is $\text{Choose}(10,000, 2) = 49,995,000$, where $\text{Choose}(n, k)$ is the number of ways to select k items from n items. The number of possible item-sets of size 9 is $\text{Choose}(10,000, 9) = 2,745,826,321,280,434,929,668,521,390,000$, which is ... well, a lot.

There are several clever algorithms to efficiently extract frequent item-sets from a list of transactions, including the Apriori, Eclat, and FP-growth algorithms. Each algorithm has many variations. The demo program uses a variation of the Apriori algorithm, which is illustrated in **Figure 2**.

Code download available at msdn.microsoft.com/magazine/msdnmag0114.

Briefly, the Apriori algorithm starts with all frequent item-sets of size $k = 1$ —individual items that meet the support threshold in the list of transactions. Then the construction process iteratively adds frequent item-sets of size $k = 2, 3$ and so on until no new frequent item-sets are found.

The image in **Figure 2** shows how item-sets of size $k = 4$ are constructed. A single list of frequent item-sets of all sizes is maintained. In the figure, there are currently three frequent item-sets with size $k = 3$: (0, 2, 3), (0, 2, 8) and (0, 3, 6). The algorithm also maintains a list of items that are valid at any given point in time. Here there are five valid items: {0, 2, 3, 6, 8}. Valid items to construct frequent item-sets of size $k = 4$ are the distinct items in all frequent item-sets of size $k-1 = 3$.

The algorithm scans each frequent item-set of size $k = 3$. For each existing item-set, a new candidate frequent item-set of size $k = 4$ is generated. So, the first candidate is (0, 2, 3, ?). The ? can be filled in with a valid item. The algorithm assumes that the items within an item-set are always stored in order, so the ? can be only 6 or 8 in this case. Each new candidate is examined to count how many times it occurs in the transactions list. If the transaction count meets the minimum support count, the candidate is added to the list of frequent item-sets.

The algorithm greatly reduces the amount of computation needed, compared to the brute-force generation approach. For example, notice that the second existing frequent item-set with size $k = 3$ is (0, 2, 8). The potential candidates will have form (0, 2, 8, ?). But because no valid item is greater than 8, there are no possible candidates generated.

Overall Program Structure

The overall structure of the demo program, with some WriteLine statements removed and a few minor edits, is presented in **Figure 3**. To create the demo, I launched Visual Studio 2012 and created a new console application program named FreqItemSets. The demo has no significant .NET dependencies, and any relatively recent version of Visual Studio will work fine. After the template code loaded into the editor, in the Solution Explorer window I renamed file Program.cs to FreqItemSetProgram.cs, and Visual Studio automatically renamed class Program for me.

At the top of the source code I deleted all unnecessary references to .NET namespaces, leaving just System and Collections.Generic. The demo begins by setting up a collection of all items in a hypothetical supermarket:

```
string[] rawItems = new string[] { "apples", "bread ", "celery",
    "donuts", "eggs ", "flour ", "grapes", "honey ", "icing ",
    "jelly ", "kale ", "lettuce " };
int N = rawItems.Length; // total items in inventory
```

Next the program sets up 10 hard-coded transactions. In most scenarios, your transactions will be stored in a text file or SQL database. Notice duplicate transactions are allowed and not all items in inventory are necessarily in a transaction:

```
string[][] rawTransactions = new string[10][];
rawTransactions[0] = new string[] { "apples", "bread ", "celery", "flour " };
rawTransactions[1] = new string[] { "bread ", "eggs ", "flour " };
...
rawTransactions[9] = new string[] { "apples", "bread ", "celery", "flour " };

After displaying the raw transactions, the demo sets up a List of 0-based
encoded transactions. Instead of hard-coding, in most situations you'd
programmatically create the numeric transactions from the raw trans-
actions. Notice that the items in each transaction are distinct and sorted:

List<int[]> transactions = new List<int[]>();
transactions.Add(new int[] { 0, 1, 2, 5 });
transactions.Add(new int[] { 1, 4, 5 });
...
transactions.Add(new int[] { 0, 1, 2, 5 });
```

```
file:///C:/FreqItemSets/bin/Debug/FreqItemSets.EXE

Begin frequent item-set extraction demo

Raw transactions are:
-----
[0] : apples bread celery flour
[1] : bread eggs flour
[2] : apples bread donuts eggs
[3] : celery donuts flour grapes
[4] : donuts eggs
[5] : donuts eggs jelly
[6] : apples bread donuts icing
[7] : bread grapes honey
[8] : apples bread celery flour kale
[9] : apples bread celery flour

Encoded transactions are:
-----
[0] : 0 1 2 5
[1] : 1 4 5
[2] : 0 1 3 4
[3] : 2 3 5 6
[4] : 3 4
[5] : 3 4 9
[6] : 0 1 3 8
[7] : 1 6 7
[8] : 0 1 2 5 10
[9] : 0 1 2 5

Setting minimum frequent support percent = 0.30
Setting minimum frequent item-set length = 2
Setting maximum frequent item-set length = 4
Using Apriori algorithm to construct frequent item-sets

Frequent item-sets in numeric form are:
< 0 1 > ct = 5
< 0 2 > ct = 3
< 0 5 > ct = 3
< 1 2 > ct = 3
< 1 5 > ct = 4
< 2 5 > ct = 4
< 3 4 > ct = 3
< 0 1 2 > ct = 3
< 0 1 5 > ct = 3
< 0 2 5 > ct = 3
< 1 2 5 > ct = 3
< 0 1 2 5 > ct = 3

Frequent item-sets in string form are:
apples bread
apples celery
apples flour
bread celery
bread flour
celery flour
donuts eggs
apples bread celery
apples bread flour
apples celery flour
bread celery flour
apples bread celery flour

End frequent item-set extraction demo
```

Figure 1 Extracting Frequent Item-Sets from Transactions

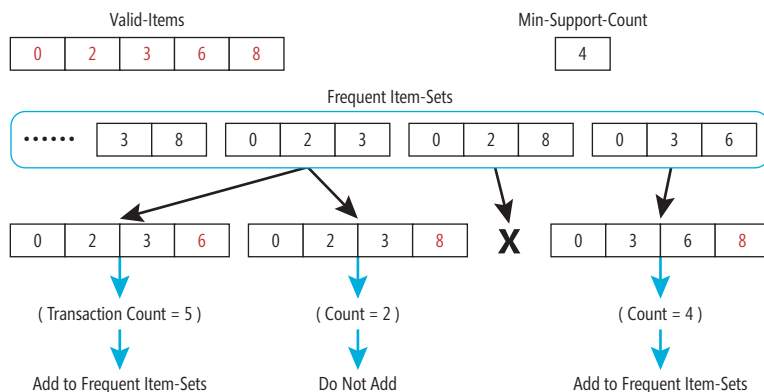


Figure 2 The Apriori Algorithm in Action

After displaying the transactions list, the three input parameter value are set up. Here, the maximum size of a frequent item-set is set to 4. You may want to scan the transactions list and set the value to the length of the longest transaction:

Figure 3 Demo Program Structure

```
using System;
using System.Collections.Generic;
namespace FreqItemSets
{
    class FreqItemSetProgram
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("\nBegin frequent item-set extraction demo\n");

                string[] rawItems = new string[] { "apples", "bread ", "celery", "donuts",
                    "eggs ", "flour ", "grapes", "honey ", "icing ", "jelly ",
                    "kale ", "lettuce" };

                int N = rawItems.Length; // total number of items to deal with ( [0..11] )

                string[][] rawTransactions = new string[10][];
                rawTransactions[0] = new string[] { "apples", "bread ", "celery",
                    "flour " };
                rawTransactions[1] = new string[] { "bread ", "eggs ", "flour " };
                rawTransactions[2] = new string[] { "apples", "bread ", "donuts",
                    "eggs " };
                rawTransactions[3] = new string[] { "celery", "donuts", "flour ",
                    "grapes" };
                rawTransactions[4] = new string[] { "donuts", "eggs " };
                rawTransactions[5] = new string[] { "donuts", "eggs ", "jelly " };
                rawTransactions[6] = new string[] { "apples", "bread ", "donuts",
                    "icing " };
                rawTransactions[7] = new string[] { "bread ", "grapes", "honey " };
                rawTransactions[8] = new string[] { "apples", "bread ", "celery",
                    "flour ", "kale " };
                rawTransactions[9] = new string[] { "apples", "bread ", "celery",
                    "flour " };

                for (int i = 0; i < rawTransactions.Length; ++i) {
                    Console.WriteLine("[" + i + "]: ");
                    for (int j = 0; j < rawTransactions[i].Length; ++j)
                        Console.WriteLine(rawTransactions[i][j] + " ");
                    Console.WriteLine("");
                }

                List<int[]> transactions = new List<int[]>();
                transactions.Add(new int[] { 0, 1, 2, 5 });
                transactions.Add(new int[] { 1, 4, 5 });
                transactions.Add(new int[] { 0, 1, 3, 4 });
                transactions.Add(new int[] { 2, 3, 5, 6 });
                transactions.Add(new int[] { 3, 4 });
                transactions.Add(new int[] { 3, 4, 9 });
                transactions.Add(new int[] { 0, 1, 3, 8 });
            }
        }
    }
}
```

```
double minSupportPct = 0.30;
int minItemSetLength = 2;
int maxItemSetLength = 4;
```

All the work is performed by a call to method `GetFrequentItemSets`:

```
List<ItemSet> frequentItemSets =
    GetFrequentItemSets(N, transactions, minSupportPct,
        minItemSetLength, maxItemSetLength);
```

Notice the return result uses a program-defined `ItemSet` class. The demo concludes by displaying the frequent item-sets in both numeric and string form.

The ItemSet Class

In essence, an item-set is just an array of integers, so no program-defined class is necessary. But, in my opinion, using a class in this case simplifies the code. The `ItemSet` class is defined in Figure 4.

Member field `N` is the total number of items in inventory. Field `k` is the size of the item-set. Array `data` holds the item values. Field `hashCode` is a unique integer to identify the item-set so duplicate

```
transactions.Add(new int[] { 1, 6, 7 });
transactions.Add(new int[] { 0, 1, 2, 5, 10 });
transactions.Add(new int[] { 0, 1, 2, 5 });

for (int i = 0; i < transactions.Count; ++i) {
    Console.WriteLine("[" + i + "]: ");
    for (int j = 0; j < transactions[i].Length; ++j)
        Console.WriteLine(transactions[i][j].ToString() + " ");
    Console.WriteLine("");
}

double minSupportPct = 0.30;
int minItemSetLength = 2;
int maxItemSetLength = 4;

List<ItemSet> frequentItemSets =
    GetFrequentItemSets(N, transactions, minSupportPct,
        minItemSetLength, maxItemSetLength);

Console.WriteLine("\nFrequent item-sets in numeric form are:");
for (int i = 0; i < frequentItemSets.Count; ++i)
    Console.WriteLine(frequentItemSets[i].ToString());

Console.WriteLine("\nFrequent item-sets in string form are:");
for (int i = 0; i < frequentItemSets.Count; ++i) {
    for (int j = 0; j < frequentItemSets[i].data.Length; ++j) {
        int v = frequentItemSets[i].data[j];
        Console.WriteLine(rawItems[v] + " ");
    }
    Console.WriteLine("");
}

Console.WriteLine("\nEnd frequent item-set extraction demo\n");
Console.ReadLine();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message); Console.ReadLine();
}
}

static List<ItemSet> GetFrequentItemSets(int N, List<int[]> transactions,
    double minSupportPct, int minItemSetLength, int maxItemSetLength) { .. }
static int CountTimesInTransactions(ItemSet itemSet,
    List<int[]> transactions) { .. }

public class ItemSet { .. }
} // ns
```


item-sets can be avoided. Field `ct` is the number of times the item-set appears in the transactions list. The `ItemSet` constructor is defined:

```
public ItemSet(int N, int[] items, int ct)
{
    this.N = N;
    this.k = items.Length;
    this.data = new int[this.k];
    Array.Copy(items, this.data, items.Length);
    this.hashValue = ComputeHashValue(items);
    this.ct = ct;
}
```

The helper method to compute the hash value is:

```
private static int ComputeHashValue(int[] data)
{
    int value = 0;
    int multiplier = 1;
    for (int i = 0; i < data.Length; ++i) {
        value = value + (data[i] * multiplier);
        multiplier = multiplier * 10;
    }
    return value;
}
```

The helper converts the item values to a single integer in reverse. For example, if the items are (0, 2, 5), the hash value is 520. The method works in reverse to deal with leading 0-items, because otherwise (0, 2, 5) and (2, 5) would both hash to 25.

Method `IsSubsetOf` returns true if the item-set object is a subset of a transaction:

```
public bool IsSubsetOf(int[] trans)
{
    int foundIdx = -1;
    for (int j = 0; j < this.data.Length; ++j) {
        foundIdx = IndexOf(trans, this.data[j], foundIdx + 1);
        if (foundIdx == -1) return false;
    }
    return true;
}
```

The method is short but slightly subtle. Because transactions and item-sets are ordered, after one item value of an item-set has been found inside a transaction, the search for the next item value does not have to begin at index 0—it can start at the next index following the location where the previous item value was found. Helper `IndexOf` is defined:

```
private static int IndexOf(int[] array, int item, int startIdx)
{
    for (int i = startIdx; i < array.Length; ++i) {
        if (i > item) return -1; // i is past target loc
        if (array[i] == item) return i;
    }
    return -1;
}
```

Method `IndexOf` also takes advantage of ordering. If the current search index is larger than the target item value being searched for, the search has gone too far and will never find the item. For example, suppose a transaction is (0, 2, 4, 5, 6, 8) and the target item being

searched for is 3. Because transactions are ordered, the very latest value 3 can occur would be in a transaction that has form (0, 1, 2, 3, x, x). The `ToString` method uses string concatenation for simplicity:

```
public override string ToString()
{
    string s = "{ ";
    for (int i = 0; i < data.Length; ++i)
        s += data[i] + " ";
    return s + "} " + ct + " ";
}
```

The GetFrequentItemSets Method

The definition of method `GetFrequentItemSets` begins:

```
static List<ItemSet> GetFrequentItemSets(int N, List<int[]> transactions,
    double minSupportPct, int minItemSetLength, int maxItemSetLength)
{
    int minSupportCount = (int)(transactions.Count * minSupportPct);
    ...
}
```

Instead of specifying an input parameter for the support threshold as a percentage of transactions, you may want to use an absolute count. Next, three important collections are instantiated:

```
Dictionary<int, bool> frequentDict = new Dictionary<int, bool>();
List<ItemSet> frequentList = new List<ItemSet>();
List<int> validItems = new List<int>();
...
```

Collection `frequentList` holds all frequent item-sets found. Instead of storing frequent item-sets of all sizes in a single list, an important alternative is to use an array of lists where there are separate lists for each item-set size. Collection `frequentDict` holds the IDs of those item-sets that have been added to `frequentList` so duplicates can be avoided. Collection `validItems` holds item values that, at any given point in the algorithm, can be added to an existing frequent item-set of size `k-1` to generate a candidate frequent item-set of size `k`. Next, the individual item values in the transaction list are counted:

```
int[] counts = new int[N];
for (int i = 0; i < transactions.Count; ++i)
{
    for (int j = 0; j < transactions[i].Length; ++j) {
        int v = transactions[i][j];
        ++counts[v];
    }
}
...
```

Then those item values that meet the minimum support count are used to create frequent `ItemSet` objects of size `k = 1`, which are then added to the list of frequent items:

```
for (int i = 0; i < counts.Length; ++i)
{
    if (counts[i] >= minSupportCount) {
        validItems.Add(i); // i is the item-value
        int[] d = new int[1]; // ItemSet ctor wants an array
        d[0] = i;
        ItemSet ci = new ItemSet(N, d, 1); // size 1, ct 1
        frequentList.Add(ci); // it's frequent
        frequentDict.Add(ci.hashValue, true); // record
    } // else skip this item
}
...
```

The main processing loop, in pseudo-code, is:

```
loop for size k = 2, 3, until done
    foreach existing freq item-set of size k-1
        foreach valid item
            create a candidate item-set
            count times candidate in transactions
            if candidate meets support threshold
                if candidate not already in frequent list
                    add candidate to frequent, rec list
            end each valid item
            update valid items
        end each freq item-set
    end main loop
```

Figure 4 The `ItemSet` Class

```
public class ItemSet
{
    public int N; // data items are [0..N-1]
    public int k; // number of items
    public int[] data; // ex: [0 2 5]
    public int hashValue; // "0 2 5" -> 520 (for hashing)
    public int ct; // num times this occurs in transactions

    public ItemSet(int N, int[] items, int ct) { . . . }
    private static int ComputeHashValue(int[] data) { . . . }
    public override string ToString() { . . . }
    public bool IsSubsetOf(int[] larger) { . . . }
    private static int IndexOf(int[] array, int item, int startIdx) { . . . }
}
```

The main processing loop is set up like so:

```
bool done = false;
for (int k = 2; k <= maxItemSetLength && done == false; ++k)
{
    done = true;
    int numFreq = frequentList.Count;
    ...
}
```

The main loop will exit when all specified sizes have been examined, or when no new item-sets of the current size are found. Because all frequent item-sets are stored in a single list (and that list is added to), the initial size of the list is stored for use by the inner loops, which are set up like so:

```
for (int i = 0; i < numFreq; ++i)
{
    if (frequentList[i].k != k - 1) continue;

    for (int j = 0; j < validItems.Count; ++j)
    {
        int[] newData = new int[k]; // data for a candidate item-set
        ...
    }
}
```

Two important features of the algorithm are that the algorithm uses only frequent item-sets from the previous iteration to construct new candidate item-sets and it examines only valid item values to complete the candidates. The candidate frequent item-sets are created:

```
for (int p = 0; p < k - 1; ++p)
    newData[p] = frequentList[i].data[p];

if (validItems[j] <= newData[k - 2]) continue;
newData[k - 1] = validItems[j];
ItemSet ci = new ItemSet(N, newData, -1);
...
```

This code is mildly tricky. First, data from the current frequent item-set is copied into the candidate. The candidate is completed with the current valid item value and, as described earlier, candidates may be eliminated based on the ordering property of item-sets. The ItemSet constructor accepts a dummy value of -1 for

Figure 5 Tying up the Two Inner Loops

```
...
if (frequentDict.ContainsKey(ci.hashValue) == true)
    continue;
int ct = CountTimesInTransactions(ci, transactions);
if (ct >= minSupportCount)
{
    ci.ct = ct;
    frequentList.Add(ci);
    frequentDict.Add(ci.hashValue, true);
    done = false;
}
} // j
} // i
...
```

Figure 6 Updating the List of Valid Items

```
...
validItems.Clear();
Dictionary<int, bool> validDict = new Dictionary<int, bool>();
for (int idx = 0; idx < frequentList.Count; ++idx) {
    if (frequentList[idx].k != k) continue;
    for (int j = 0; j < frequentList[idx].data.Length; ++j) {
        int v = frequentList[idx].data[j]; // item
        if (validDict.ContainsKey(v) == false) {
            validItems.Add(v);
            validDict.Add(v, true);
        }
    }
}
validItems.Sort();
} // next k
...
```

the ct field, because the number of times the candidate appears in the transactions is not yet known.

The two inner loops are tied up with the code shown in **Figure 5**.

If the candidate already appears in the frequent item-set list, there's no need to analyze it further. If not, and if the candidate meets the minimum support count, the candidate is a winner and it's added to the list of frequent item-sets. Boolean done tracks whether or not any new item-sets have been found. For any given value of k, if no new frequent item-sets are found, there's no possibility that a frequent item-set will ever be found.

After all candidates for the current size k have been constructed and examined, the list of valid items for the next iteration is updated, as shown in **Figure 6**.

Although not entirely obvious at first, it turns out that when constructing new candidate frequent item-sets of size k using existing frequent item-sets of size k-1, the only valid items to complete the candidates are item values that occur in the frequent item-sets of size k-1. This update process is time-consuming, and in some scenarios you'll get better performance by skipping it and instead using the original list of valid items created for size k = 1.

The method concludes by filtering the results by minimum item-set length:

```
...
List<ItemSet> result = new List<ItemSet>();
for (int i = 0; i < frequentList.Count; ++i)
{
    if (frequentList[i].k >= minItemSetLength)
        result.Add(new ItemSet(frequentList[i].N,
                                frequentList[i].data, frequentList[i].ct));
}
return result;
}
```

Helper method CountTimesInTransactions, used earlier, is defined:

```
static int CountTimesInTransactions(ItemSet itemSet,
    List<int[]> transactions)
{
    int ct = 0;
    for (int i = 0; i < transactions.Count; ++i) {
        if (itemSet.IsSubsetOf(transactions[i]) == true)
            ++ct;
    }
    return ct;
}
```

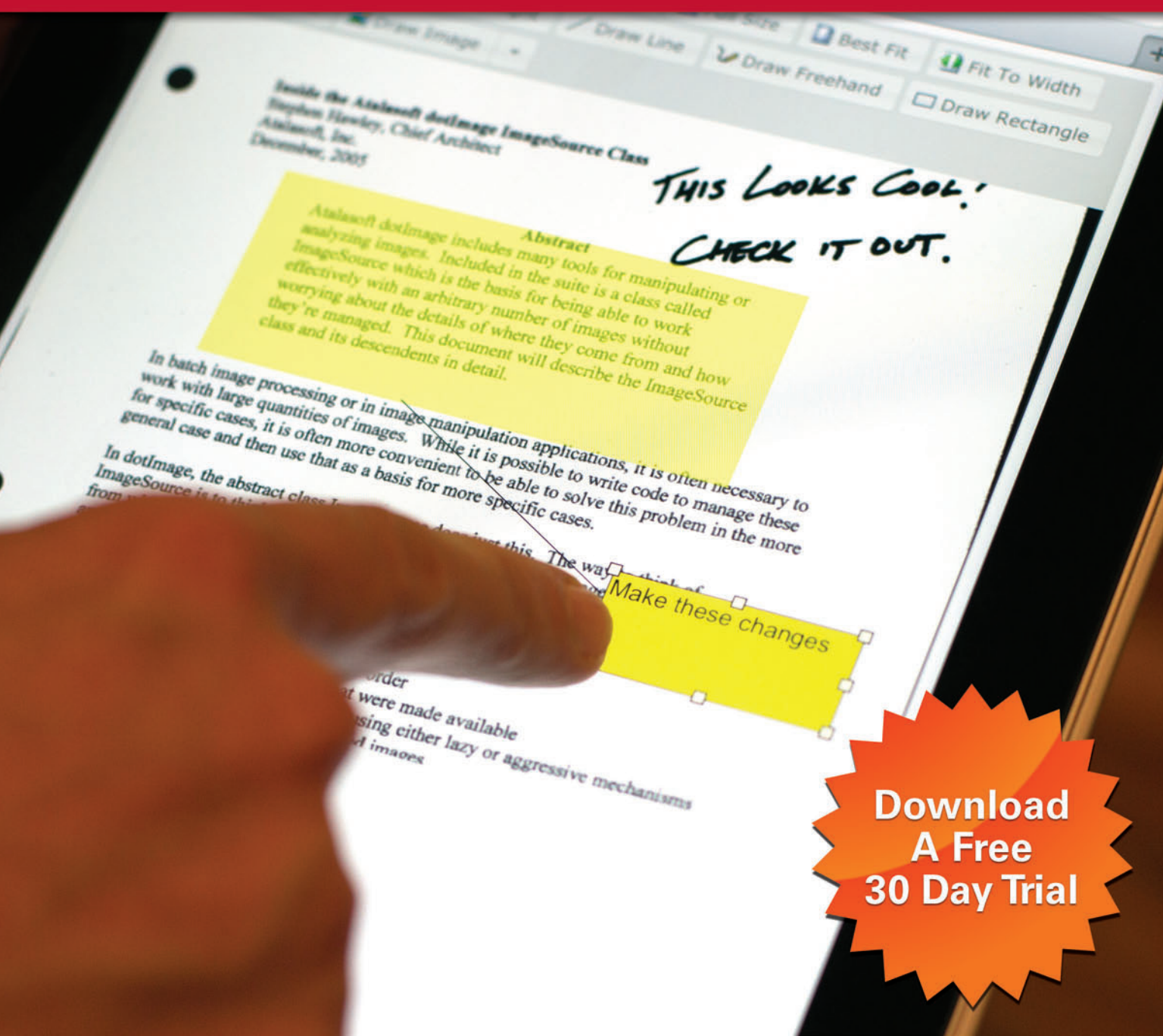
Wrapping Up

The code and explanation presented in this article should get you up and running if you ever need to determine frequent item-sets from a list of transactions using the Apriori algorithm. Although it's unlikely you'll ever be working directly with supermarket transactions, there are many scenarios where extracting frequent item-sets can be useful. In the case of NUI, you can think of transactions as a list of user commands—for example, the text typed into a search text box—and the items as the words that make up the command. Identifying those words that frequently occur together can help generate better search results. ■

DR. JAMES McCaffrey works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products including Internet Explorer and Bing. He can be reached at jammc@microsoft.com.

THANKS to the following technical expert for reviewing this article:
Richard Hughes (Microsoft Research)

Build A Mobile Document Viewer with Annotations, Touch Interfaces, Zooming, Pagination, and More



Download
A Free
30 Day Trial





Getting Started with Oak: Database Interaction

Welcome back. I've been walking through getting started with Oak, a dynamic approach to Web development that embodies ideas from the Ruby and Node.js worlds while still retaining the C# language, the ASP.NET MVC framework and the rest of the Microsoft .NET Framework you enjoy. The essence of Oak is “fast feedback, frictionless development and low ceremony,” in the words of project creator Amir Rajan.

When I left off, Oak was complaining it couldn't connect to a database. In this column, I'll show you how to do the following: wire up Oak to a SQL Server instance, add another related type to the system (comments on blog entries), build the database in Oak, relate the two types together using Oak, and see how Oak and its build system handle database interaction.

When We Left Our Hero ...

The last time Oak was running, it popped up an error message and helpful explanatory text, an excerpt of which is shown in **Figure 1**. (As a reminder, if you were following along in the last column—at msdn.microsoft.com/magazine/dn532208—you'll need to kick off the server and sidekick runner.) Before I go much further, take note of how Oak isn't just displaying an exception stack trace and leaving it to me to discover or research what could be causing the problem. It's actually trying to diagnose the problem and suggesting potential solutions: in this case, “Update the web config with your server instance.”

Sure enough, a quick glance at the web.config file I picked up from the “warmup” gem shows that a simple placeholder is there, pointing to a “(local)” data source, which might work for a lot of configurations, depending on how the local SQL Server instance is configured. That said, it's as trivial to drop in a LocalDB connection string as it is to drop in a remote SQL Server string (or a Windows Azure SQL Database, if it's desirable to keep data in the cloud). I like using LocalDB, which is usually just a connection string of “Server=(localdb)\v11.0;Integrated Security=true” for explorations like this, because it's small and cleans up fairly easily. Whichever SQL connection you use, just plug it into the <configuration>/<connectionStrings>/<add> element and refresh the page. In fact, Oak has a rake (the Ruby build

tool) task to do this across the codebase for you: “rake update_db_server[(localdb)\v11.0]” (the backslash needs to be escaped).

Uh ... Houston?

Unfortunately, when run again, Oak acts as if nothing has changed. Also unfortunately, that's entirely true. Sidekick isn't watching for changes to web.config, it seems, only source files. Considering how rarely you change the web.config file, this isn't the end of the world, but it does put a small cramp in what's otherwise a pretty effortless process. To get sidekick to recognize the project has changed, you just have to trigger a save of some important file, such as HomeController.cs or some other source file. You can do this in two ways: switch over to that file, pick a random place in the file, hit the space bar and delete that space (which convinces Visual Studio that the file is “dirty” and forces a save), or just manually kick off rake from a command prompt.

Once that's done, a browser refresh brings up a different error (“Invalid object name 'Blogs'”), and you're squarely into the world of database migrations.

Data, Data, Data!

Like its conceptual predecessors, Oak wants to manage databases and database schema for you, so the database can remain more or less “hidden” from your view. In this case, Oak doesn't want to just “automagically” craft a database out of thin air, because you'll probably have some opinions about how the schema should look—and even if you don't, the database admins often do. (Who wins that battle, or who *should* win that battle, is a discussion best had over beers—preferably long after the project has shipped.)

In Oak, seeding the database is done using a particular controller, the SeedController, found nested right next to the HomeController in SeedController.cs. That file contains a definition for the SeedController already, but—more importantly for your purposes—it also contains a Schema class, which will facilitate

Update the web config with your server instance

It doesn't look like I could connect to your SQL Server instance. Update the web.config to point to a SQL Server instance. Here is

```
data source=(local); initial catalog=Blog; integrated security=true
```

Figure 1 The Oak Project Help Window After an Error Is Thrown

Figure 2 The Schema Class

```
public class Schema
{
    // This is the method you'll want to alter
    public IEnumerable<Func<dynamic>> Scripts()
    {
        // Replace all content inside of the Scripts() method with this line
        yield return CreateBlogsTable; // Return just the pointer to the function
    }

    public string CreateBlogsTable() // Here is the function definition
    {
        // This is an example, your table name may be different
        // For more information on schema generation check out the Oak wiki
        return Seed.CreateTable("Blogs",
            Seed.Id(),
            new { Name = "nvarchar(255)" },
            new { Body = "nvarchar(max)" }
        );
    }

    public void SampleEntries()
    {
    }

    public Seed Seed { get; set; }

    public Schema(Seed seed) { Seed = seed; }
}
```

the steps necessary to build the schema and, optionally, put some sample data in place. Bear in mind, by the way, that the default convention of Oak is that objects are stored in a database table of pluralized name, so Blog objects will be stored in a table named Blogs. **Figure 2** shows the Schema class.

Note the strange use of “yield return” in the Scripts method. For those of you who could never figure out what this did in C# 2.0, “yield return” creates an anonymous “stream” of objects and hands back an `IEnumerable<T>` pointing to that stream. In this particular case, it’s a stream of functions, and in this particular scenario, it’s a stream of one function (`CreateBlogsTable`). In essence, you’re setting up a stream of actions (`Func<dynamic>` instances) representing how to create the database, and Oak will take each action or function handed back from this stream and execute it. This way, when a new change to the schema comes into play, that change can be captured in a new function (“`CreateComments-`

Figure 3 Describing the One-to-Many Relationship Between Blogs and Comments

```
public class Blog : DynamicModel
{
    Blogs blogs = new Blogs();

    // Define comments
    Comments comments = new Comments();

    public Blog() { }

    public Blog(object dto) : base(dto) { }

    // Add an Associates method to add the Comments() method
    IEnumerable<dynamic> Associates()
    {
        // And define the association
        // For other examples of associations check out the Oak wiki
        yield return new HasMany(comments);
    }
}
```

Table,” if you will) and simply added to the list. If desired, you could “version” the database schema by calling the first function `Version1`, the second one `Version2` and so on. The Oak wiki has more information on database migrations at bit.ly/1bgods5.

(For those who recall my “Multiparadigmatic .NET” series, which begins at msdn.microsoft.com/magazine/ff955611, yes, this is a pretty functional-oriented way of approaching this problem.)

The `Seed.Id` function creates the canonical primary key column: an auto-incrementing integer value marked as a primary key. The Oak wiki (bit.ly/1iKfclb) contains a reference on creating a database using the `Seed` class to do so, but there’s always the ad hoc fallback of doing straight SQL if desired:

```
public IEnumerable<string> AdHocChange()
{
    var reader = "select * from SampleTable".ExecuteReader();

    while (reader.Read())
    {
        // Do stuff here like yield return strings
    }

    var name = "select top 1 name from sysobjects".ExecuteScalar() as string;

    yield return "drop table SampleTable";

    yield return "drop table AnotherSampleTable";
}
```

Add the `AdHocChange` method as another “yield return” method to call, and Oak will merrily carry out these commands as well. (Note that if you lose track of that wiki link, Oak includes it in the error/help message it displays.)

I Cannot Make Bricks Without Clay!

By the way, if the database needs some seed data, this is also part of the `SeedController`’s responsibility, and again it falls to the `Schema` class to do it, this time via the `SampleEntries` method. Because this system doesn’t have any real seed data that needs to be here, I’ll leave this alone.

Once the `SeedController` is compiled, `sidekick` will redeploy the project, but like most controllers it won’t get activated until hit with an HTTP request. If you didn’t glance over the `SeedController`, take a quick look. It exports four POST endpoints: `PurgeDB`, `Exports`, `All` and `SampleEntries`. While you could certainly hit the endpoints yourself, this is the kind of repetitive task that’s best left to automation—which in the case of Oak means rake. Sure enough, “rake reset” will drop all the tables and regenerate the schema (it does a POST to `/seed/PurgeDB` and then another POST to `/seed/all`), and “rake sample” will drop the tables, regenerate the schema and generate the sample data (POST `/seed/SampleEntries`). Just for completeness, by the way, a “rake export” (or a POST `/seed/export`) will return the SQL statements used to do those things.

(Curl or Ruby can make doing a POST from the command line a one-line exercise, by the way. The `Rakefile.rb` file has examples of how to do the POST using `Net::HTTP::post_from` that are pretty easy to cut and paste into another `.rb` file, if you don’t want it buried inside the `Rakefile`. Or you could use this as a gateway drug toward learning Ruby. Nobody’s judging.)

It’s Alive!

Assuming there are no typos, once rake reset is done, a refresh in the browser brings up a working Web page with a simple text field

(for the blog title) and a “submit” button. Entering a new blog title will generate an error, however—although Blogs exist, there’s sort of an assumption (based on what’s in the Index.cshtml view) that there’s also some kind of a thing associated with Blogs called Comments.

In the simple relational model that dominates the blogging engine world, blogs have a one-to-many relationship to comments. I want to model that in this system as well, so first I need a data type for Comment objects, which is brain-dead simple, again. In fact, because Comments are (like Blog objects) essentially dynamic objects, with no interesting elements to them, they don’t even need a model class definition—the stock all-dynamic object (the Gemini type, for those who remember my August 2013 column, “Going Dynamic with the Gemini Library,” at msdn.microsoft.com/magazine/dn342877) is fine.

(And yes, if you haven’t had a “we’re a long way from Kansas, Toto” kind of moment before now, this is definitely time to stop and ponder: You’re working with a business object whose type you never bother to define.)

To describe the relationship of Comments to Blogs, however, you have to do two things. First, Comments needs a repository (just as Blogs did in my last column):

```
public class Comments : DynamicRepository
{
}
```

Figure 4 A Database Description of the Comments Table

```
public class Schema
{
    public IEnumerable<Func<dynamic>> Scripts()
    {
        yield return CreateBlogsTable;
        yield return CreateCommentsTable;
    }

    public string CreateBlogsTable() // Here is the function definition
    {
        return Seed.CreateTable("Blogs",
            Seed.Id(),
            new { Name = "nvarchar(255)" },
            new { Body = "nvarchar(max)" }
        );
    }

    public string CreateCommentsTable() // Here is the function definition
    {
        return Seed.CreateTable("Comments",
            Seed.Id(),
            new { BlogId = "int", ForeignKey = "Blogs(Id)" },
            new { Body = "nvarchar(max)" }
        );
    }
}
```

Figure 5 Adding Blog Entries

```
public void SampleEntries()
{
    var blog1 = new // Store the ID
    {
        Name = "Hello, Oak Blog",
        Body = "Lorem ipsum dolor sit amet, consectetur adipiscing elit."
    }.InsertInto("Blogs");
    new { Body = "great job!", BlogId = blog1 }.InsertInto("Comments");

    new
    {
        Name = "Here are cat pictures!",
        Body = "Meowem hisum collar sit amet, addipisces lick."
    }.InsertInto("Blogs");
}
```

More significantly, the Blog class needs to be revamped slightly to capture the one-to-many relationship, both directly (Blog owning a collection of Comment objects, which in this case means having a Comments object as a field), and indirectly (so Oak knows that Blog and Comment objects are connected in the database, and how). This is done by introducing a new method, Associates, that describes the relationship, as shown in **Figure 3**.

As you can see, it’s not really that far off from what I said you needed to do: The model matches the abstract concept of a blog pretty closely. This is where the beauty and power of using dynamic techniques can be seen. This one change (the Associates method doing a “yield return” of a HasMany) actually triggers three new methods being added to Blog—Comments, CommentIds and NewComment—to support the relationship of Comment objects to Blog objects, all of which are pure scaffolding and would normally require you to write those methods “the hard way” were you using normal, non-dynamic C#. Naturally, though, for all of this to work against the database, you need to back it up with a database description of the comment table, which means you’re back to SeedController (and a subsequent rake reset), as shown in **Figure 4**.

Because I’m here, by the way, I’ll add a couple of blog entries, just to show off how to use the SampleEntries method. It’s a lot easier than you might think, as shown in **Figure 5**.

Once again, the use of dynamic objects and extension methods makes it almost not look like C# anymore. (In this case, you’re not creating a dynamic object so much as creating an instance of an anonymous object with the autogenerated properties Title and Body, and then using the extension method InsertInto to do the actual insertion.) And, by the way, the only reason to trap the object in the blog1 local variable is so it can be used as the blog ID value for the comment on it.

Go ahead. Refresh the database with a rake sample and then refresh the browser.

Next: User Input Validation

This is turning into some interesting stuff, if it wasn’t already. You have no model types defined, yet you have a working model and database storage for it. No SQL scripts were involved (though it’s fair to suggest that repeated use of ad hoc methods in the Schema class could easily turn into SQL scripts), and it’s important to point out that all of the code doing all of this is (still) tucked away in source form in the Oak folder in the scaffolded project, in case you need to debug or just feel like browsing.

There are still a few things yet to do, such as validating the user input to make sure it’s all good and correct before storing, but that’s a good subject for next time.

Happy coding! ■

TED NEWARD is the principal of Neward & Associates LLC. He has written more than 100 articles and authored and coauthored a dozen books, including “Professional F# 2.0” (Wrox, 2010). He’s an F# MVP and speaks at conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com if you’re interested in having him come work with your team, or read his blog at blogs.tedneward.com.

THANKS to the following technical expert for reviewing this article:
Amir Rajan (Oak project creator)



ALMFORUMSEATTLE

expertise worth sharing

Washington State Convention Center • April 1-3, 2014

Register by February 15th to save \$300

follow us on



keynote speakers

Scott Ambler

ALM Thought Leader and Co-creator of DAD Framework



Disciplined Agile Delivery: The Foundation for Scaling Agile

Steve Denning

Award-winning Author



Transforming Management through Agile

Ken Schwaber

Industry Legend and Co-Creator of Scrum



The State of Agile

Sam Guckenheimer

Product Owner, Microsoft Visual Studio



Transforming software development in a world of devices and services

plenary speakers

Mike Brittain

Director of Engineering, Etsy



Principles and Practices of Continuous Deployment

James Whittaker

Distinguished Technical Evangelist, Microsoft



Intent Commerce

Dave West

Chief Product Officer, Tasktop



Lean ALM

sponsors



diamond



platinum





The Steam Drill

*John Henry said to the captain, "A man ain't nothin' but a man.
But before I let your steam drill beat me down,
I'll die with a hammer in my hand."*

—"The Ballad of John Henry," American traditional

John Henry occupies a mighty place in American folklore (bit.ly/3HqIMd). Some folks say he's just a tall tale, but others insist he's based on a real man. John Henry "drove steel" in the late 1800s, pounding a sledgehammer onto a steel drill rod, boring holes into rock for blasting railroad tunnels. When a steam-powered drill threatened his job, John Henry challenged it to a duel—and won. "John Henry drove 16 feet and the steam drill only made nine," the ballad says. But no matter which version of the song you hear, it always ends badly: "He worked so hard, he broke his poor heart, so he laid down his hammer and he died."

Every advance in computing has
generated a new abstraction
layer. And with every higher level
of abstraction, the programmers
who strove mightily to master
the lower level get angry.

What does such a quintessential hardware guy have to do with software, I hear you wondering. I had my own John Henry moment last week, teaching threads in my "Programming Microsoft .NET" class at Harvard Extension. I started my students with raw threads, then showed them the thread pool, then worked into the design problems of synchronization and thread affinity. I concluded with a brutal homework assignment that tempted them to use the thread pool but imposed requirements that didn't allow it. (Student: "Platt, you're a sadistic bastard." Me: "Um, yeah, what's your point?")

The next week I showed them the Task Parallel Library (TPL), with its `Parallel.For` and `Parallel.ForEach` constructs. I then told them I myself didn't trust it. It was too new. And it didn't save me enough time and effort to compensate for the loss of control and transparency. Then I realized what I dinosaur I'd become.

Every advance in computing has generated a new abstraction layer. And with every higher level of abstraction, the programmers who strove mightily to master the lower level get angry. Now any idiot can do what they busted their asses to learn. Not fair!

For example, when those magnetic disk thingies first came out, programmers had to command the hardware directly—take head 3, move to track 19, wait for sector 247 to come around and then read it. Then some smart guy figured out how to hide the hardware behind a software abstraction layer that provided a logically contiguous series of bytes called a file.

"It's way inefficient," the fogies screamed. "You kids these days have no respect for resources. Think of all those sectors being wasted by, what do you call them, directories?"

"You bunch of old farts," sneered the kids. "Do you still program with only ones and zeroes?"

When did I leap from upstart to fogey? I have no problem with the thread pool. I show students how it monitors the CPU usage of its threads, assigning more threads to the job when some of them block. But at least there I can see the threads, as opposed to the TPL, where they're hidden.

I've always found there's no substitute for knowing what's going on under the hood. Abstractions always leak. Disks become fragmented, scattering files over inconvenient clusters that require wasteful movement of metal. The students who did the best in my MFC class years ago were the ones who heeded my advice to study Charles Petzold's classic book on the Windows API. Whenever I fix a customer's threading-related problem, it always—and I mean every single time, without exception—stems from a fundamental misunderstanding of what a thread is and does.

So go ahead, use the TPL `Parallel.For` and `Parallel.ForEach`. But if you use them as a crutch without understanding how they really work, it's like using an inner tube to paddle out into the middle of Boston Harbor without knowing how to swim. You'll be screaming for the Coast Guard, for guys like me, to come rescue you. Fortunately, we're on the ball, because we know we have to be. A geek ain't nothin' but a geek. And before I let your TPL beat me down, I'll die with a thread handle in my hand. ■

DAVID S. PLATT teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.

Building Blocks for Global Data Quality Success



A strong foundation for enterprise data starts with Melissa Data.

Our powerful, scalable data cleansing and integration tools help you profile, cleanse, consolidate, and enrich your contact data. Gain a better understanding of your customer, vendor and supplier data; improve deliverability; increase cost savings; and enhance your operational efficiencies with Melissa Data.

Advanced Functionalities:

- Verify, correct, and enhance addresses for 240+ countries
- Add lat/long coordinates to addresses all over the world
- Match and consolidate data to create the golden record
- Append missing contact data like phone numbers and email addresses
- Integrate into many technologies with multiplatform capabilities

www.MelissaData.com
or call 1-800-MELISSA (635-4772)

MELISSA DATA®
Your Partner in Data Quality

WE HELP YOU DELIVER STUNNING CLIENT-SIDE APPLICATIONS



Essential Studio for **JavaScript**

All the client-side enterprise controls you need are in one suite, from one company.

- ★ Over 40 components—Most comprehensive JavaScript library in the market.
- ★ Exclusive OLAP grid and chart—Visualize business intelligence data on the web.
- ★ Powerful chart—Visualize data in ways you didn't think possible on the client side.
- ★ Enhanced grid—Employ a rich set of features, such as Microsoft Excel-like grouping.
- ★ Stunning maps and gauges—Build client-side dashboards with ease.
- ★ Controls work on any platform—No IIS or specific back end required.

Deliver innovation with ease.

Download a free, 30-day evaluation today.

syncfusion.com/javascript

+1 888-9-DOTNET

